

AD-A041 459

COMPUTER SCIENCES CORP FALLS CHURCH VA

F/G 9/2

A COMPREHENSIVE DATA BASE ACCESS METHODOLOGIES DESIGN GUIDE.(U)

SEP 76 J EMORY, S GREEN, R GRIMES, O HASLOP

DCA100-75-C-0029

UNCLASSIFIED

CSC-R493700019-2-2

CCTC-TM-123-76

NL

1 OF 4
AD
A041459



**C
C
T
C**

ADA 041 459

AD NO. _____
DDC FILE COPY

**DEFENSE
COMMUNICATIONS
AGENCY**



TECHNICAL MEMORANDUM
TM 123-76
28 SEPTEMBER 1976



**COMMAND
& CONTROL
TECHNICAL
CENTER**

**A COMPREHENSIVE
DATA BASE ACCESS
METHODOLOGIES
DESIGN GUIDE**

TECHNICAL MEMORANDUM

Q D D C
RECEIVED
JUL 12 1977
D

DISTRIBUTION STATEMENT A

Approved for public release
Distribution Unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TM 123-76	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Comprehensive Data Base Access Methodologies Design Guide.		5. TYPE OF REPORT & PERIOD COVERED Final rept.
7. AUTHOR(s) John/Emory, Otis/Haslop Sally/Green, Dr. Anupam/Shah Robert/Grimes, Hans Ullmann		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Corporation Falls Church, VA 22046		8. CONTRACT OR GRANT NUMBER(s) DCA 100-75-C-0029 new
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Communications Agency Washington, D.C. 20305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Director Command and Control Technical Center Pentagon, Rm BE685 ATTN: C422 Washington, D.C. 20301		12. REPORT DATE September 28, 1976
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		13. NUMBER OF PAGES
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
18. SUPPLEMENTARY NOTES		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Data base management, data structures, stacks, queues, dequeues, sequential, indexed sequential, plex, trees, bit maps, chains, rings, pointers, inverted, hashing, indexing, data base access, relational, normal form, data integrity, networks, distributed data, data base security.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The Comprehensive Data Base Access Methodologies Design Guide is produced to assist programmers, analysts, and data base managers in the application of state-of-the-art methodology in the use and design of data structures, access mechanisms, and management of computer processed data. The scope of the document covers the principal methods in general use today. The material has been organized for easy comprehension by relatively inexperienced personnel and provides a source for further perusal of related literature.		

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

REPORT TO CONGRESS

1. TITLE (Include Project Number, if any)

2. AUTHOR (Name, Address, and Organization)

3. PERIODICITY (Frequency of Publication)

4. DATE OF PUBLICATION

5. DATE OF REPORT

6. DATE OF REVISION

7. DATE OF REVIEW

8. DATE OF APPROVAL

9. DATE OF DISTRIBUTION

10. DATE OF DESTRUCTION

11. DATE OF RECLASSIFICATION

12. DATE OF DECLASSIFICATION

13. DATE OF REVIEW

14. DATE OF APPROVAL

15. DATE OF DISTRIBUTION

16. DATE OF DESTRUCTION

17. DATE OF RECLASSIFICATION

18. DATE OF DECLASSIFICATION

19. DATE OF REVIEW

20. DATE OF APPROVAL

21. DATE OF DISTRIBUTION

22. DATE OF DESTRUCTION

23. DATE OF RECLASSIFICATION

24. DATE OF DECLASSIFICATION

25. DATE OF REVIEW

26. DATE OF APPROVAL

27. DATE OF DISTRIBUTION

28. DATE OF DESTRUCTION

29. DATE OF RECLASSIFICATION

30. DATE OF DECLASSIFICATION

31. DATE OF REVIEW

32. DATE OF APPROVAL

33. DATE OF DISTRIBUTION

34. DATE OF DESTRUCTION

35. DATE OF RECLASSIFICATION

36. DATE OF DECLASSIFICATION

37. DATE OF REVIEW

38. DATE OF APPROVAL

39. DATE OF DISTRIBUTION

40. DATE OF DESTRUCTION

41. DATE OF RECLASSIFICATION

42. DATE OF DECLASSIFICATION

43. DATE OF REVIEW

44. DATE OF APPROVAL

45. DATE OF DISTRIBUTION

46. DATE OF DESTRUCTION

47. DATE OF RECLASSIFICATION

48. DATE OF DECLASSIFICATION

49. DATE OF REVIEW

50. DATE OF APPROVAL

51. DATE OF DISTRIBUTION

52. DATE OF DESTRUCTION

53. DATE OF RECLASSIFICATION

54. DATE OF DECLASSIFICATION

55. DATE OF REVIEW

56. DATE OF APPROVAL

57. DATE OF DISTRIBUTION

58. DATE OF DESTRUCTION

59. DATE OF RECLASSIFICATION

60. DATE OF DECLASSIFICATION

61. DATE OF REVIEW

62. DATE OF APPROVAL

63. DATE OF DISTRIBUTION

64. DATE OF DESTRUCTION

65. DATE OF RECLASSIFICATION

66. DATE OF DECLASSIFICATION

67. DATE OF REVIEW

68. DATE OF APPROVAL

69. DATE OF DISTRIBUTION

70. DATE OF DESTRUCTION

71. DATE OF RECLASSIFICATION

72. DATE OF DECLASSIFICATION

73. DATE OF REVIEW

74. DATE OF APPROVAL

75. DATE OF DISTRIBUTION

76. DATE OF DESTRUCTION

77. DATE OF RECLASSIFICATION

78. DATE OF DECLASSIFICATION

79. DATE OF REVIEW

80. DATE OF APPROVAL

81. DATE OF DISTRIBUTION

82. DATE OF DESTRUCTION

83. DATE OF RECLASSIFICATION

84. DATE OF DECLASSIFICATION

85. DATE OF REVIEW

86. DATE OF APPROVAL

87. DATE OF DISTRIBUTION

88. DATE OF DESTRUCTION

89. DATE OF RECLASSIFICATION

90. DATE OF DECLASSIFICATION

91. DATE OF REVIEW

92. DATE OF APPROVAL

93. DATE OF DISTRIBUTION

94. DATE OF DESTRUCTION

95. DATE OF RECLASSIFICATION

96. DATE OF DECLASSIFICATION

97. DATE OF REVIEW

98. DATE OF APPROVAL

99. DATE OF DISTRIBUTION

100. DATE OF DESTRUCTION

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	

COMMAND AND CONTROL TECHNICAL CENTER

Technical Memorandum TM 123-76

September 28, 1976

A COMPREHENSIVE DATA BASE
ACCESS METHODOLOGIES DESIGN GUIDE

Submitted by:

R. A. Marion
R. A. Marion
Project Officer

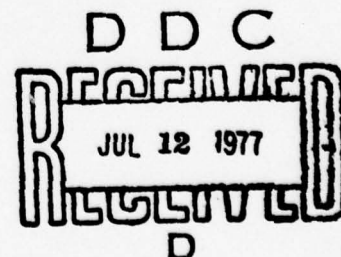
Reviewed by:

M. E. Champagn
Mr. M. E. Champagn
Chief, Development Division

Approved by:

J. A. Painter
Dr. J. A. Painter
Technical Director

Distribution of this document is unlimited. This document is prepared by Computer Sciences Corporation for the Director, Command and Control Technical Center, WWMCCS ADP Directorate, the Pentagon, Washington, D.C. 20301.



A Comprehensive Data Base
Access Methodologies Design Guide

by

John Emory
Sally Green
Robert Grimes
Otis Haslop
Dr. Anupam Shah
Hans Ullmann

Prepared for the

Command and Control Technical Center
WWMCCS ADP Directorate
of the
Defense Communications Agency
Washington, D. C.

under

Contract DCA100-75-C-0029

Teleprocessing Software Development Operation
Systems Division

Computer Sciences Corporation
Falls Church, Virginia 22046

Report R493700019-2-2

September 28, 1976

CONTENTS

	<u>Page</u>
Abstract	xiii
<u>Chapter 1 - Organization of Systems</u>	1-1
1.1 General	1-1
1.2 Programming Interface	1-1
1.2.1 Languages	1-1
1.2.2 Technical Specialists	1-5
1.2.3 Integration	1-6
1.3 Data Concepts	1-8
1.3.1 Enterprise	1-8
1.3.2 Information	1-13
1.3.3 Logical	1-16
1.3.4 Physical	1-19
<u>Chapter 2 - Hardware</u>	2-1
2.1 General	2-1
2.2 Definition of Terms	2-1
2.2.1 Central Processing Unit	2-1
2.2.2 Input/Output Devices	2-1
2.2.3 Secondary Storage	2-4
2.3 Timing	2-14
2.4 Capacity	2-15
2.5 Cost	2-16
2.6 Device Evaluation	2-16

PART I

<u>Chapter 3 - Serial Structures</u>	3-1
3.1 General	3-1
3.2 Physical Ordering	3-1
3.3 Logical Ordering	3-4
<u>Chapter 4 - Sequential Access of Serial Structures</u>	4-1
4.1 General	4-1
4.1.1 Insertions/Deletions	4-1
4.1.2 Flagging	4-1
4.1.3 End-of-Space Condition	4-1
4.1.4 Element Formats	4-2

<u>Chapter 4 (Cont'd)</u>	<u>Page</u>
4.2 Access Methodology	4-2
4.2.1 Serial Methods	4-2
4.2.2 Sequential Methods	4-7
4.3 Access Considerations	4-7
4.4 Advantages	4-8
4.5 Disadvantages	4-8
4.6 Variations	4-9
4.6.1 Blocking	4-9
4.6.2 File Limits	4-12
4.7 Applications	4-12

<u>Chapter 5 - Random Access of Serial Structure</u>	5-1
--	-----

5.1 General	5-1
5.1.1 Insertions/Deletions	5-1
5.1.2 Flagging	5-1
5.1.3 End-of-Space Condition	5-1
5.1.4 Element Formats	5-2
5.2 Access Methodology	5-2
5.2.1 Relative Sequence Method	5-2
5.2.2 Defined Key Method	5-4
5.2.3 Indexed Sequential Method	5-8
5.2.4 Inverted List File	5-17
5.2.5 Access by Secondary Key	5-19
5.3 Access Considerations	5-19
5.4 Advantages	5-20
5.5 Disadvantages	5-20
5.6 Variations	5-21
5.7 Applications	5-21

PART II

<u>Chapter 6 - Random Structures</u>	6-1
6.1 General	6-1
6.2 Logical Structures	6-1
6.2.1 Representation Standards	6-1
6.2.2 Tree Structures	6-9
6.2.3 Plex Structures (Networks)	6-20

<u>Chapter 6 (Cont'd)</u>		<u>Page</u>
6.3	Physical Structures	6-25
6.3.1	Pointers	6-26
6.3.2	Chains and Rings	6-28
6.3.3	Indexing	6-38
6.3.4	Bit Maps	6-46
6.4	Advantages	6-50
6.4.1	Trees	6-50
6.4.2	Plexes (Intersecting Chains and Rings)	6-50
6.4.3	Pointers	6-50
6.4.4	Chains and Rings	6-50
6.4.5	Indexing	6-51
6.4.6	Bit Maps	6-51
6.5	Disadvantages	6-51
6.5.1	Trees	6-51
6.5.2	Plexes	6-51
6.5.3	Pointers	6-52
6.5.4	Chains and Rings	6-52
6.5.5	Indexing	6-52
6.5.6	Bit Maps	6-52
6.6	Applications	6-52
<u>Chapter 7 - Random Access of Random Structures</u>		<u>7-1</u>
7.1	General	7-1
7.2	Access Methodology	7-1
7.2.1	Direct Address	7-1
7.2.2	Dictionary Look-Up	7-2
7.2.3	Hashing	7-2
7.2.4	Indexed Sequential Access	7-10
7.3	Insertions/Deletions	7-10
7.4	Access Considerations	7-11
7.5	Advantages	7-11
7.6	Disadvantages	7-12
7.7	Variations	7-12

<u>PART III</u>		<u>Page</u>
<u>Chapter 8 - Introduction to Relational Structures</u>		8-1
8.1	General	8-1
8.2	Essential Elements.	8-3
8.2.1	Relational Data Base.	8-3
8.3	Sublanguages	8-8
8.3.1	Language Characteristics	8-8
8.3.2	Language Approach.	8-9
8.4	Access Considerations	8-19
8.5	Advantages	8-20
8.6	Disadvantages	8-20
8.7	Applications	8-20
<u>Chapter 9 - Normalization of Relations</u>		9-1
9.1	General	9-1
9.2	Normalization	9-1
9.2.1	First Normal Form (1NF)	9-2
9.2.2	Second Normal Form (2NF)	9-9
9.2.3	Third Normal Form	9-9
<u>PART IV</u>		
<u>Chapter 10 - Integrity</u>		10-1
10.1	General	10-1
10.1.1	Reliability.	10-1
10.1.2	Validity	10-1
10.2	Loss of Integrity	10-1
10.2.1	Hardware Failure.	10-2
10.2.2	Software Failure	10-2
10.3	Maintaining Integrity.	10-2
10.3.1	Dump	10-2
10.3.2	Journal.	10-3
10.3.3	Recovery	10-4
10.3.4	Detection	10-5
10.3.5	Data Independence	10-6
10.4	Application Considerations.	10-7
10.4.1	Batch Processing.	10-7

<u>Chapter 10 (Cont'd)</u>		<u>Page</u>
10.4.2	On-Line Systems	10-7
10.4.3	Shared Data Base	10-8
 <u>Chapter 11 - Security</u>		 11-1
11.1	General	11-1
11-2	Definitions	11-1
11.2.1	Security	11-1
11.2.2	Privacy	11-2
11.2.3	Integrity	11-2
11.3	Fundamental Characteristics	11-2
11.3.1	Functional	11-2
11.3.2	Inexpensive	11-3
11.3.3	Simple	11-3
11.3.4	Program Generality	11-3
11.4	Mathematical Model	11-3
11.5	Security Compromises	11-3
11.5.1	Causes	11-4
11.5.2	Specific Hardware/Software Examples	11-10
11.6	Methods to Maintain Security	11-14
11.6.1	Hardware Checks	11-14
11.6.2	Data Locks	11-15
11.6.3	Passwords	11-16
11.6.4	Encryption	11-18
11.6.5	Combination of Methods	11-19
11.6.6	Monitoring and Surveillance	11-19
11.7	Other Security Considerations	11-21
11.7.1	Recovery Plan	11-21
11.7.2	Distributed Data Bases	11-21
11.8	Applications	11-21
11.8.1	DBMS Proposed by DBTG	11-22
11.8.2	Integrated Data Store	11-22
11.8.3	World Wide Data Management System	11-23
 <u>Chapter 12 - Networks and Distributed Data</u>		 12-1
12.1	General	12-1
12.2	Networks	12-1
12.2.1	Components	12-2

<u>Chapter 12 (Cont'd)</u>	<u>Page</u>
12.2.2 Organization	12-19
12.3 Distributed Data Base	12-21
12.3.1 General	12-21
12.3.2 Environment	12-21
12.3.3 Concept	12-21
12.3.4 Creating a Distributed Data Base	12-24
12.3.5 Considerations	12-24
12.3.6 File Segmentation	12-25
12.3.7 DDB File Access Methods	12-34
12.3.8 DDB Reliability and Integrity	12-41
12.4 Applications	12-42
12.4.1 Multiplexed Information and Computing Service (Multics) ...	12-42
12.4.2 Advanced Research Projects Agency Network (ARPANET)...	12-45
12.4.3 Prototype World-Wide Military Command and Control System (WWMCCS) Intercomputer Network (PWIN).....	12-48
12.4.4 Distributed Computer System (DCS).....	12-51
12.4.5 ALOHA Network	12-52
12.4.6 TELEX Switching System	12-53
12.4.7 AUTODIN II	12-54
 <u>Chapter 13 - Abbreviations and Acronyms</u>	 13-1
<u>Chapter 14 - Glossary</u>	14-1
<u>References</u>	R-1
<u>Appendix A - Mathematical Model</u>	A-1
<u>Index</u>	X-1

ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1-1	Self-Contained Systems	1-3
1-2	Host Language Systems	1-4
1-3	Views of Technical Specialists	1-7
1-4	Views of Information	1-7
1-5	Data Management System	1-9
1-6	Language Interfaces	1-10
1-7	Table of Terms	1-12
1-8	Examples of Attributes	1-12
1-9	An Information System	1-14
1-10	Overall Schema with Separate Subschemas	1-17
1-11	Relation of Physical Elements	1-21
2-1	Input/Output Devices	2-3
2-2	Nine-Track Tape	2-5
2-3	Multi-File Tape	2-6
2-4	Drum	2-8
2-5	Disc	2-10
2-6	Data Cell Device	2-13
2-7	Terabit Memory System	2-13
3-1	Serial Structures Physically Ordered Through Application of Algorithmic Techniques	3-2
3-2	Serial Structure Physically Ordered Through Sorting on a Defined Key Field Imbedded in Data Elements	3-5
3-3	Serial Structures Logically Ordered Through Application of Algorithmic Techniques	3-6
3-4	Serial Structure Logically Ordered Through Sorting on a Defined Key Field Imbedded in Data Elements Through Appended Pointers	3-11
4-1	Flow Diagram of a Sequential Access Mechanism Designed to Operate on Physically Ordered Stacks	4-3
4-2	Flow Diagram of a Sequential Access Mechanism Designed to Operate on Physically Ordered Queues	4-5
4-3	Data Flow When Serially Accessing Serial Data Structures Using Algorithmic Techniques	4-6
4-4	Insertion for a Physically Ordered Serial Structure of Fixed Length Elements Blocked Into Logically Ordered Records With Three Logical Data Element Capacity	4-11
4-5	File Limits Clause as Provided by COBOL	4-13
4-6	Chain-Order Sorted	4-17
4-7	Chain-Order Sorted Within Type	4-17

<u>Figure</u>		<u>Page</u>
5-1	Illustrative Example of Binary Searching a Physically Ordered Linear List of Data Elements	5-6
5-2	Levels of Index	5-11
5-3	Block Search for Key K_S	5-14
5-4	Inverted List File Organization.	5-18
6-1	Example of Difference Between Physical and Local Data Organization.	6-2
6-2	Mapping Relations	6-4
6-3	I-D-S Symbology	6-7
6-4	Three Trees	6-11
6-5	Unbalanced Trees.	6-13
6-6	Binary Trees.	6-14
6-7	Binary Representation of Forest.	6-15
6-8	B-Tree Examples.	6-18
6-9	Four-Digit Trie	6-19
6-10	Examples of Plex Structures	6-20
6-11	Plex Structure of Five Record-Types Used For a Purchasing Application	6-21
6-12	Complex Plex Structure With Only Two Record-Types	6-22
6-13	Cycle	6-23
6-14	Loop	6-23
6-15	Equivalent Plex Tree Structures	6-24
6-16	Relationship in Figure 6-12 Redrawn as Two Two-Level Tree Relationships	6-25
6-17	Chained Records	6-29
6-18	Intersecting Chains.	6-31
6-19	Chains With Two-Way Pointers.	6-32
6-20	Ring With Pointers in One Direction Only	6-34
6-21	Intersecting Rings	6-35
6-22	Two-Way Chain With Pointers in Both Directions.	6-36
6-23	Multidirectional Chains	6-39
6-24	Rings With Head Pointers	6-40
6-25	Coral Ring	6-41
6-26	Skip-Linked Chain	6-42
6-27	Skip-Linked Coral Ring	6-42
6-28	Multilist Chain.	6-43
6-29	Parallel Cellular Chains	6-43
6-30	Basic Index	6-45
6-31	Bit Map Relations.	6-47
6-32	Bit Vectors	6-48

<u>Figure</u>		<u>Page</u>
7-1	Folding.	7-4
7-2	Shifting.	7-4
7-3	Chain-Order Sorted Within Type, With Additional Pointers	7-13
7-4	I-D-S Structure Containing CALC Chains MAKE, TYPE, and COLOR	7-15
8-1	Current Trend in System Architecture	8-2
8-2	Relational Versus Standard Data Terminology	8-4
8-3	Departments and Jobs Data Model in Relational Form with Two "Constant" Relations	8-10
8-4	Results of Four Queries on Relational Model (Figure 8-3) . . .	8-11
8-5	Two Sample Projections Resulting in R_1 and R_2 , Respectively	8-13
8-6	Sample Projection Resulting in R_3	8-14
8-7	Two Sample Joins Resulting in R_4 and R_5 , Respectively . . .	8-15
8-8	Sample Division Resulting in R_6	8-16
8-9	Sample Projection with Record Selection	8-17
8-10	Sample Complex Example Resulting in R_8	8-17
8-11	Relational System Summaries.	8-21
9-1	Relation of Relations : Unnormalized	9-2
9-2	RDIR Partially Normalized	9-3
9-3	RDIR in First Normal Form.	9-4
9-4	Departments as a Repeating Group within Jobs	9-4
9-5	Elimination of Repeating Group in Figure 9-4	9-5
9-6	Plex Entry Schema	9-6
9-7	Proliferation of Cross-Referencing Links.	9-7
9-8	Elimination of Plex Structure in Figure 9-6	9-8
9-9	Cross-Referencing Structure	9-8
9-10	Relations in Second Normal Form.	9-10
9-11	A Relation not in Third Normal Form	9-10
9-12	Relations in Third Normal Form.	9-11
9-13	RDIR in Third Normal Form	9-12
11-1	Typical Threats	11-5
11-2	Potential Threats and Associated Costs	11-5
11-3	Risk Analysis	11-6
11-4	Domain Structure	11-12
11-5	DBTG Implementation of Security [D].	11-22
11-6	Implementation of Security in I-D-S [O]	11-23
11-7	Privacy Clause [P]	11-23

Figure

12-1	Data Traffic Types.	12-11
12-2	Protocol Categories.	12-12
12-3	Protocol Layering.	12-14
12-4	Protocol Appendage Cycle.	12-15
12-5	Network Organization.	12-22
12-6	Single Copy.	12-27
12-7	Single Copy With Passive Backup.	12-27
12-8	Single Copy With Active Backup.	12-29
12-9	Single Primary Copy With Partial Copies.	12-30
12-10	Multiple Primary Copies.	12-32
12-11	Typical MULTICS Directory Hierarchy.	12-43
12-12	ARPA Network, Geographic Map (June 1975).	12-46
12-13	ARPA Network, Logical Map (June 1975).	12-47

ABSTRACT

One of the recent advances in computer network technology has been the development and implementation of the Prototype WWMCCS Intercomputer Network (PWIN) facility. PWIN provides program-program communications, interactive terminal access, workload sharing, teleconferencing, and stand-alone terminal access, all in a package designed for full utilization of the entire system by one or more network subscribers. This package will require network users at all levels to know the best procedures for storing and retrieving data distributed over the network to reduce both inter- and intra-host processing overhead.

This Comprehensive Data Base Access Methodologies Design Guide is produced to assist network programmers, analysts, and data base managers in the application of state-of-the-art methodology in the use and design of data structures and access mechanisms, and in the management of network data.

The scope of this document covers the principal methods in general use today. The material has been organized for easy comprehension for relatively inexperienced personnel and provides a source for further perusal of current literature. Since PWIN is an ongoing project, this document will be updated on the basis of new developments in the field.

The references include selected recent publications dealing with the organization and access of data as well as some of the more standard texts on the subject. All were used liberally in the preparation of this document. Because the information has been extensively structured, and considerable liberty has been taken to reconcile differences in terminology, few direct acknowledgments of sources are included in the text. However, to aid the reader, the first three listed were considered most useful.

CHAPTER 1 - ORGANIZATION OF SYSTEMS

1.1 GENERAL

The complexity of today's information systems requires more than a casual investigation and planning of organized structures. These complexities have led to the development of multi-talented teams involved in designing, developing, and utilizing the data. The evolution of new specializations has produced specific skill categories in the technical staff such as systems analysts, data base administrators, and application programmers. Division of technical responsibility provides not only a better approach to effective utilization of resources, but provides a new perspective to associations within the information structure. The definition of these associations is the purpose of this chapter.

1.2 PROGRAMMING INTERFACE

1.2.1 Languages

The main objective of the user is to obtain the desired information from the conglomerate data existing within the computing system. The request for information can be applied to a computer system at one of three separate levels. The most basic level, the Input/Output Control System (IOCS), provides the interface with the channel and device controls utilized by the user. Also at this level, some error checking, access techniques, and data integrity controls are available to the user. There is typically very little device independence within the IOCS. The next higher level of sophistication is the Data Base Management System (DBMS).^{*} The software at this level provides a more convenient interface to the IOCS and also furnishes extended capabilities. The DBMS assumes more responsibility for data integrity and allows the collection of data into cataloged areas. Most DBMSs provide a considerable degree of device independence. The highest level of system software present today is the Data Management System (DMS). The DMS includes all the capabilities of a DBMS

^{*} The definitions of DBMS and DMS are frequently found to be reversed in the literature; however, the above definitions are compatible with current WWMCCS usage.

and eliminates any redundancy of data between various applications. The sharing of data allows more efficient utilization of storage resources, thus providing more economical organization of information. The DMS contains two main areas, the interpreter and the data specifications processor. The interpreter processes the requests from the user and coordinates the data specifications with the request to provide access to the data. There exists an external directory capable of storing information necessary to describe the specifications of the entire enterprise.* For the DMS to understand both the context of the request and the organization structure of the stored data, a communication capability must exist. This capability is created through several types of computer languages. The most commonly used languages are the conventional programming languages such as COBOL, FORTRAN, PL1, etc., and assembly language. These types of languages are typically used to communicate with the computer.

As the need for communication with the DMS increased, a data base language (DBL) was devised. A DBL provides facilities for describing and manipulating data. The subsets of a DBL are: a data manipulation language (DML), a data description language (DDL) and a physical description language (PDL).

1.2.1.1 Data Manipulation Language

The main property of the Data Manipulation Language (DML), is its ability to give instructions to the DMS in a format conveying the user's need to address data. There is a two-way interplay between the user's request and the data base management system; one to present a request to address data and the second to receive/store information or recognize an indication of an error. The data manipulation languages of today's computers have either been designed as self-contained, such as TDMS, or incorporated into a host language, such as I-D-S and IMS (see Figures 1-1 and 1-2). The second alternative has been chosen by the CODASYL Data Base Task Group (DBTG) as the future standard for referencing a COBOL structure within COBOL programs. Proposed extensions of COBOL would be in the form of new methods for describing data relations and a set of call statements providing the necessary interface to the data management

*A DMS stores masses of data. It is not concerned with file structures, as is a DBMS. Data, in a DMS, is found via a directory.

IDENTIFICATION	ACRONYM	ORIGINATOR	INITIAL RELEASE
Generalized Information System	GIS	IBM	1969 September
MARK IV ¹	MARK IV	Informatics Inc.	1968 (full system)
NMCS Information Processing System	NIPS/FFS	DCA, IBM, NMCS	1968 July
Time-shared Data Management System	TDMS	System Dev. Corp.	1969
User Language/1	UL/1	RCA	1969
Data Language/1	DL/1	IBM	Dec 1973
Integrated Database Management System	IDMS	Cullinane Corp.	May 1973
System 2000	System 2000	MRI Systems	June 1970
Management Data Query System ²	MDQS	Honeywell	1974

¹ A file management system rather than a DMS.

² Commercial version of World Wide Data Management System (WWDMS) initially released in 1973.

Figure 1-1. Self-Contained Systems

IDENTIFICATION	ACRONYM	ORIGINATOR	INITIAL RELEASE
Journal of Development Common Business Oriented Language ¹	COBOL	CODASYL	1970 April (specification)
Data Base Task Group Proposal ¹	DBTG	CODASYL Programming Language Committee	1971 April (specification)
Integrated Data Store	I-D-S	Honeywell Information Systems	1963
Information Manage- ment System	IMS	IBM North American Rockwell	1969 September
System Control-1	SC-1	Western Electric and Auerbach ²	1970 July
Adaptable Data Base System	ADABAS ³	Software AG	March 1971
Total	Total	Cincom System Inc.	1969
Multiple Access and Retrieval System	MARS ³	University Computer Co.	1969
FORTE	FORTE	Burroughs	1973

- 1 While DBTG proposal specifies necessary components for a DMS, COBOL Language usually is not considered a DMS.
- 2 Developed by Western Electric with Auerbach providing technical assistance based upon DM-1 concept.
- 3 Can also be a stand-alone, however primary usage is with a host language.

Figure 1-2. Host Language Systems

system. The largest benefit derived from imbedding a DML in COBOL would be an increase in portability of COBOL programs developed on different computer hardware.

1.2.1.2 Data Description Language

In order to process a user request through the use of a DML, there must be some method for identifying the data to be addressed. This is accomplished through the Data Description Language (DDL). This language could be either an extension of a host language, or a self-contained language. The global descriptions provided by the DDL are not of general interest to the user since the user usually is only concerned with his specific data and not with the entire set of relations available to all users. The global aspects of the data and its description are the interest and responsibility of the Data Base Administrator (DBA).

1.2.1.3 Physical Description Language

Just as there needs to be a language to describe the logical organization of the data, the DDL, there must also be a similar language to relate data to the physical hardware. This is the Physical Data Description Language (PDL)*. The PDL maps the assignment of any particular piece of data on a hardware device which is part of a computer system architecture. Buffer management, paging, overflow, and address search techniques, are items in the PDL. The complexities of the PDL require the services of a highly skilled systems analyst. The PDL varies widely with data management systems and computers.

1.2.2 Technical Specialists

To define the role of each of the technical specialists involved in the development of data bases, their view of the total system must be categorized. Figure 1-3 is a presentation of these views and the relations between them. The broadest view of any system is called the description of the enterprise which is a global conception of the entire system.

To understand the operation and use of a PDL, one must be thoroughly familiar with the characteristics of the hardware and software which comprise a computing system. Since a Systems Analyst (SA) has this familiarity, a PDL is usually designed,

* Referred to as the device/media control language by the CODASYL DBTG.

developed, and maintained by SAs. They are trained to follow the distribution of data throughout the various devices to best meet user data requirements.

To perform his responsibilities, the SA must receive a description of the enterprise data model from the DBA. The DDL can be used to provide a description of the data model.

To use a DDL, one must be thoroughly familiar with the data structures, associated access or processing techniques, and associated information relating to usage, security, etc. The DBA is generally assigned the responsibility of integrating all data requirements for all users in the enterprise into the data model. He generates and maintains the data model through use of a DDL.

To perform his responsibilities, the DBA must have a description of the data submodel for each application program in the enterprise. He must, in turn, provide a complete logical description of the data model including associated information to the SA. Close cooperation between the DBA and SA is essential since data cannot be accessed until both the logical and physical components of system catalogs and directories are complete.

Although the DDL is used by both the DBA and SA, it is generally designed, developed, and maintained by SAs.

An Application Programmer (AP) is interested in specific parts of the data model only. He manipulates data through the use of a DML and a DDL. He uses the DDL to describe his view (submodel) of data for the DMS, and the DML to actually access or address data.

1.2.3 Integration

Figure 1-4 illustrates the various specialists' views of information. For convenience of exposition, the user includes ADP and non-ADP people who access data interactively.

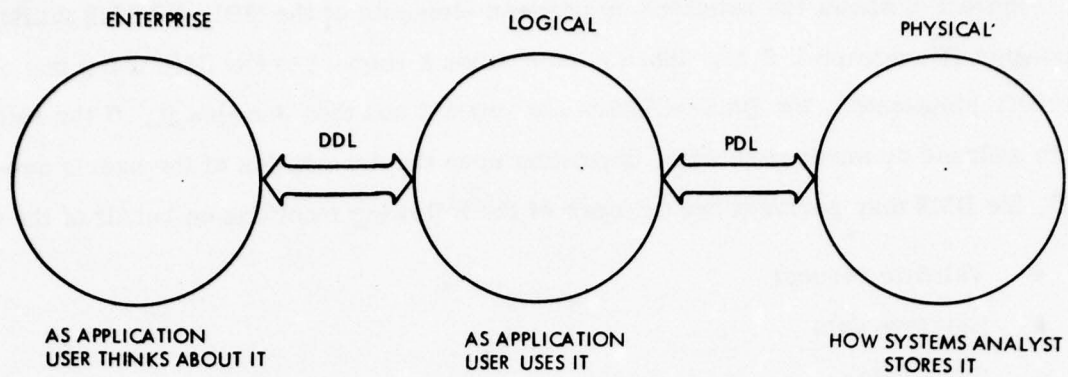


Figure 1-3. Views of Technical Specialists

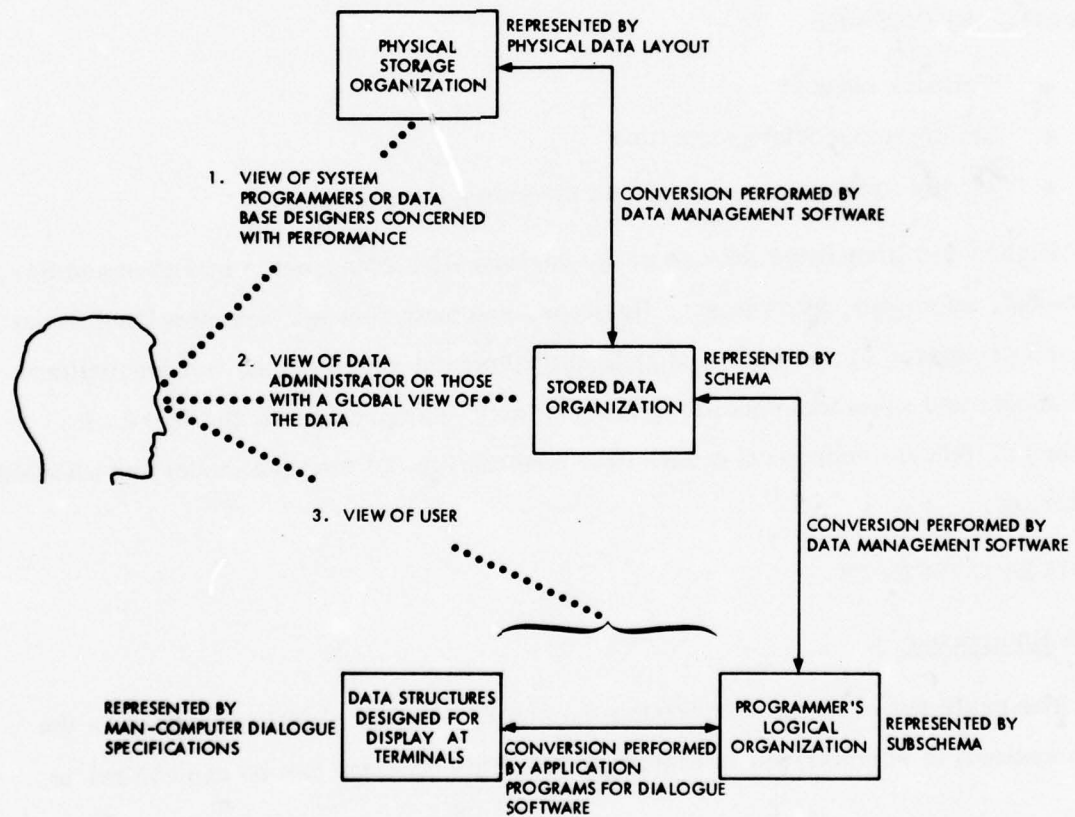


Figure 1-4. Views of Information

Figure 1-5 shows the relationship between elements of the DBL and DMS major components (Paragraph 1.2.1). When a user sends a request to the DMS using one or more DBL statements, the DMS validates the request and then decodes it. If the request asks to address or manipulate data, depending upon the description of the user's submodel, the DMS may perform one or more of the following functions on behalf of the user:

- Validate request
- Retrieve data
- Store data
- Temporarily expand description of user's submodel to accommodate the data address.

If the request specifies modification of the data model for an enterprise or a user's submodel, the DMS will:

- Validate request
- Modify appropriate structure
- Modify appropriate catalog and directory components.

Figure 1-6 illustrates the use of the various DBL components by DBAs and SAs. The model, submodel, storage specifications, and user requests are specified in the DBL and processed by the DMS. Model, and submodel requirements are equivalent to the model and submodel specifications, but may be expressed in any form. Requirements indicate coordination activities essential to defining the model and allocating the storage.

1.3 DATA CONCEPTS

1.3.1 Enterprise

The basic set of terms comprising the enterprise (the global conception of the entire system) is summarized in Figure 1-7.* This diagram can be used to relate

*The usage of several terms in this discussion, such as schema, are normally considered within the area of logical processing. However, to provide the necessary emphasis between the conceptual aspects of data and the logical organization within a computer, the separation is necessary. When investigating the references, please utilize Figure 1-7 as a translation table.

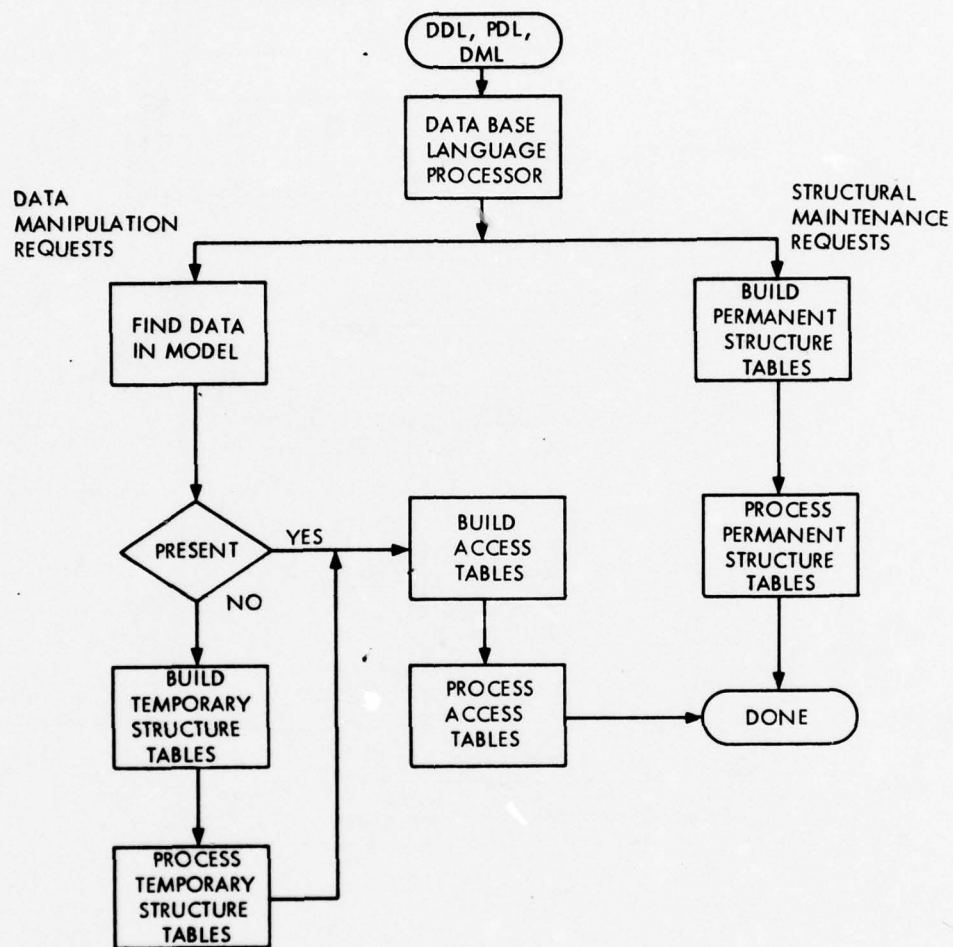
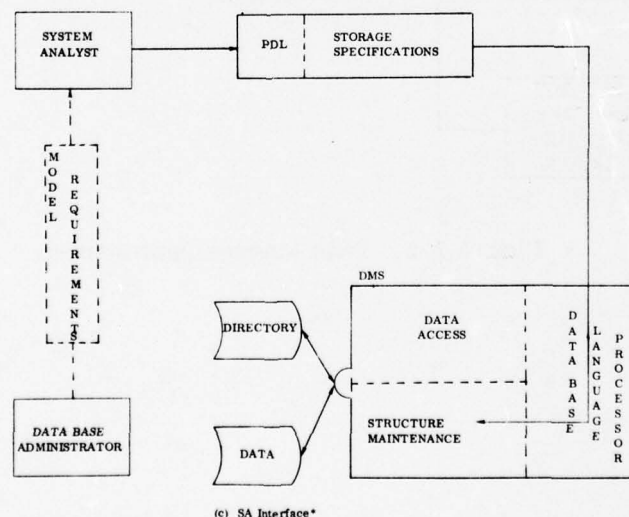
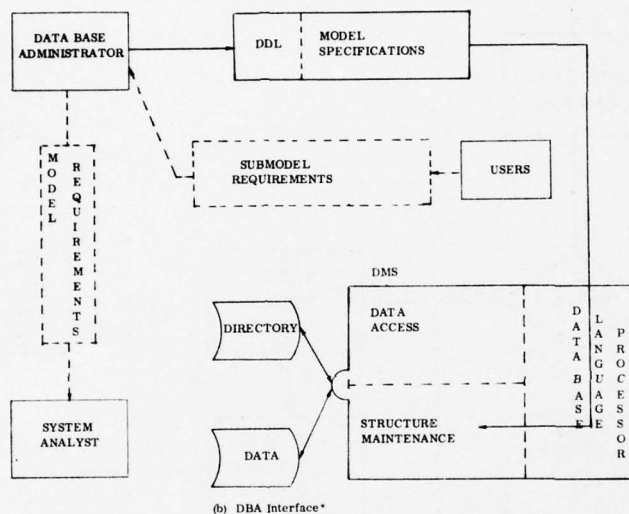
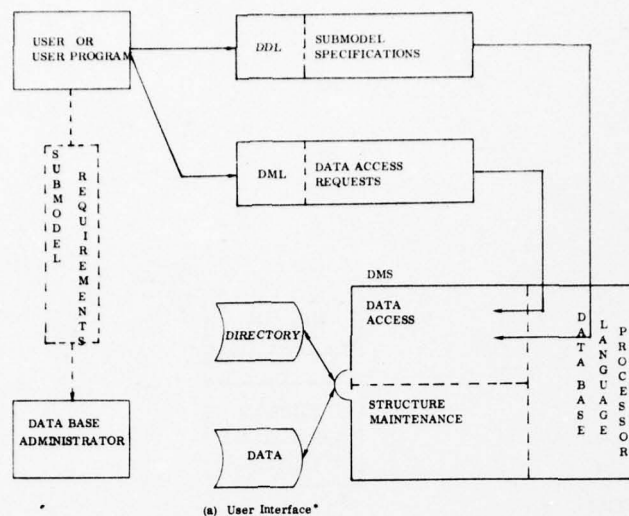


Figure 1-5. Data Management System



* Dotted lines indicate coordinating activities specified on an administrative level.

Figure 1-6. Language Interfaces

the enterprise definitions with the logical and the physical. There is a one-to-one relation for each term in the enterprise to each term in the logical; however, no similar relation exists with the physical terms. As previously explained, the DDL is needed to describe the detailed relations between the enterprise and the logical and the PDL is used to relate logical data names to physical storage addresses.

1.3.1.1 Real-World Information

The real world is a source of information which is not constrained by the limits of computer technology. From the viewpoint of data processing, however, the abstract real world is represented by the information generated by it so that this real-world information can interact with the rest of the enterprise. While the real world is comprised of physical entities with physical properties, man can only perceive information in terms of entity-sets and entity-representations described by valued attributes.

Since this discussion deals only with information, the term "entity" refers not only to the real object, but also to its informational representation as perceived by man.

1.3.1.2 Entity

An entity in the real world is an object or relation about which information is desired. Different entities are distinguished by their different properties.

An informational entity is a collection of valued attributes such as color, name, identification number, or cost. In Figure 1-8, the entity being described represents an employee. Two entities, such as employee and automobile, may have some attributes in common (i.e., weight, or color) and other attributes not in common (i.e., surname, or length of guarantee). Dissimilar entities have attributes not in common, and are distinguished by different entity-names.

1.3.1.3 Attribute

An attribute is the informational analog of a real-world property. Similar entities have all the same attributes, but are distinguished by different values of the same attribute. Attributes for the employee in Figure 1-8 include employee identification, name, sex, etc.

Unformatted			Formatted		
Real-World Information	Organized	Logical	Physical	Storage Types	Hardware Nomenclature
	Model		Data	Secondary Disc Drum Tape Mass Cartridge Primary Main Memory	Device Unit
	Schema	Base	Bank		Module
	Subschema	Set *	Set *		Volume (Reel)
Entity-set		File **	Extent		Cylinder
			Block		Track
Entity	Aggregate	Record			Sector
Attribute	Field	Item			Word
	Description				Byte
Value		Value			Bit

NOTES: *Concatenation of files to form an expanded file; hence, a physical set is the logical set of extents defining the concatenated files (as used in IBM terminology).
 **Examine text for definition. (Paragraph 1.3.3.3).

Figure 1-7. Table of Terms

Name of Attribute: Value of Attribute	Employee ID	Name	Sex	Union Status	Title	Salary	Department
	3875	John Doe	1	N-U	Administrator	2000	35
	2196	Jane Smith	0	U	Asst. Manager	2100	42
	1897	Rita Lamb	0	N-U	Sr. Manager	3000	35
	4231	George Brinks	1	U	Clerk	1475	12
	6517	Sherman Jones	1	N-U	Sr. Accountant	3521	42
	4215	John King	1	N-U	Sr. Clerk	1645	12
	0091	Robert Curtis	1	N-U	Sr. Manager	3105	42

Figure 1-8. Examples of Attributes

1.3.1.4 Entity Set

A collection of similar entities is defined as an entity set. As previously noted, similar entities are distinguished by the fact that they possess the same attributes for which statistics are to be recorded (e.g., collection of similar entities, such as employees, Figure 1-8).

1.3.2 Information

Information can be described as processed data. Input/output data, computed data, and intermediate results all provide a form of information. Many different items of information can be extracted or deduced from the same basic data. Data is converted into information after it is interpreted, in some way, by the user. In Figure 1-9 information is not only the data attached to the DBMS, but also includes the input/output transactions, other data available to the DMS, forms necessary to supply raw data, and the knowledge by which man controls the enterprise.

1.3.2.1 Organized Data

Organized data is described in terms of fields, aggregates, schemas, subschemas, and models. The organized data extracts from the total information only that portion which can be expressed through relations such as schemas and models. The organized information is further restricted in this discussion to that portion which can be computerized.

1.3.2.2 Field

A field is the smallest element of an organized structure for which information can be recorded. Just as the entity has attributes, the field likewise has descriptions which denote qualities peculiar to the particular field in question.

The description of a field describes particular types of characteristics necessary to provide the DBA the ability to control the definition of the data elements within the

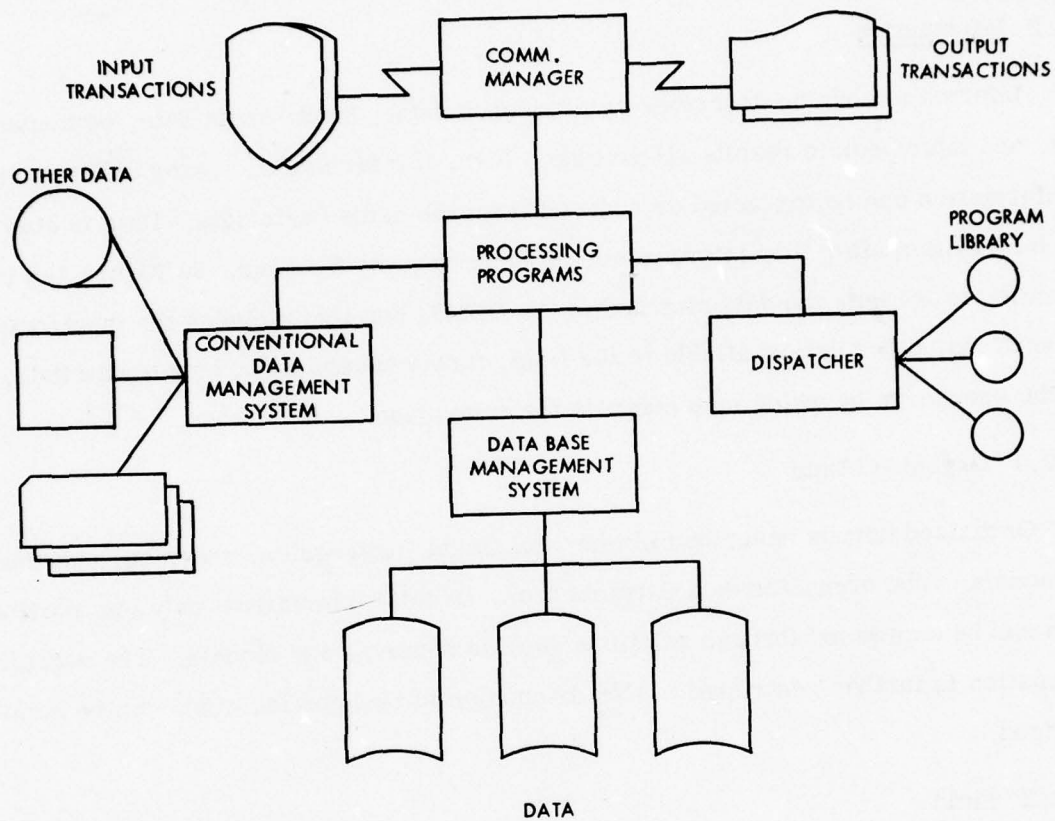


Figure 1-9. An Information System

model. Descriptions of fields include such data descriptions as length, grouping, and type of data (e.g., in a host language, such as COBOL, the main descriptions of a field are defined by the PICTURE clause).

1.3.2.3 Aggregate

The aggregate is a collection of organized data elements used to identify the entities envisioned in the real world. The aggregate may be of two types: vectors and repeating groups. A vector is a one-dimensional ordered collection of fields possessing identical characteristics. The repeating group is a collection of organized information which occurs multiple times within an entity. A repeating group may be composed of single data fields, vector aggregates, or other repeating groups.

1.3.2.4 Schema

A schema is a complete description of organized information and is constructed from fields and aggregates and includes interconnecting relations. It includes names and descriptions of all fields within the aggregates. The schema is utilized to produce an organized view of the information which is necessary to satisfy relational data requirements for all users.

The methods by which schemas can be drawn diagrammatically are varied; however, two basic formats are described in detail in Paragraph 6.2.1. The most commonly used method within the diagrams presented herein is similar to that of Figure 6-2. The other was developed by Bachman [M] to describe the specific application of the Integrated Data Store (I-D-S) language.* The success of an organized data structure is directly related to the DBA's ability to include all user requirements within the schema. The lower portion of Figure 1-10 shows a typical example of the portion of information necessary to support a parts inventory system. The DBA must envision all the relations within the information necessary to provide each AP with the information for his particular application.

*The I-D-S nomenclature is described in Paragraph 6.2.1.2. See also Reference [V].

1.3.2.5 Subschema

The subschema refers to a subset of the organized information described within a schema. The related portion of a structure which will satisfy a particular application is the subschema for said application. Figure 1-10 indicates the existence of a series of subschemas within a given schema. Each AP is cognizant of his own subschema and is not concerned with the total schema. The relations expressed by the subschema (and the schema) are not dependent on the usage of the data, only the relationship between aggregates. The submodel is the term describing the relations between particular usages of data and the organized information. The subschema should not be confused with submodel since a submodel may consist of various subschemas depending upon the operating characteristics of a given application.

1.3.2.6 Model

The model is the organized information in its totality. The model may include one or more schemas to support organization requirements for related data. The enterprise must be described within the model to assure that all users have the ability to access the data. The DDL is the vehicle by which the DBA describes the organization of all data contained within the model.

1.3.3 Logical

Logical terms present definitions of the structure as understood by the AP. The terms are constrained only by the particular DBL used to describe the relations exhibited by the schemas. Many of the terms (i.e., item, record, file) have been utilized throughout the literature to describe various aspects of logical processing. In some instances, the definitions have been modified by various authors in the contexts of specific discussions. Therefore, in Figure 1-7, a warning is given to be aware of different definitions by authors when reading the literature. To eliminate ambiguity, this manual does not include new definitions, but only points out differences in common uses. If unclear, terms will be modified by a prefix indicating a logical or physical aspect, or by an explicit comment.

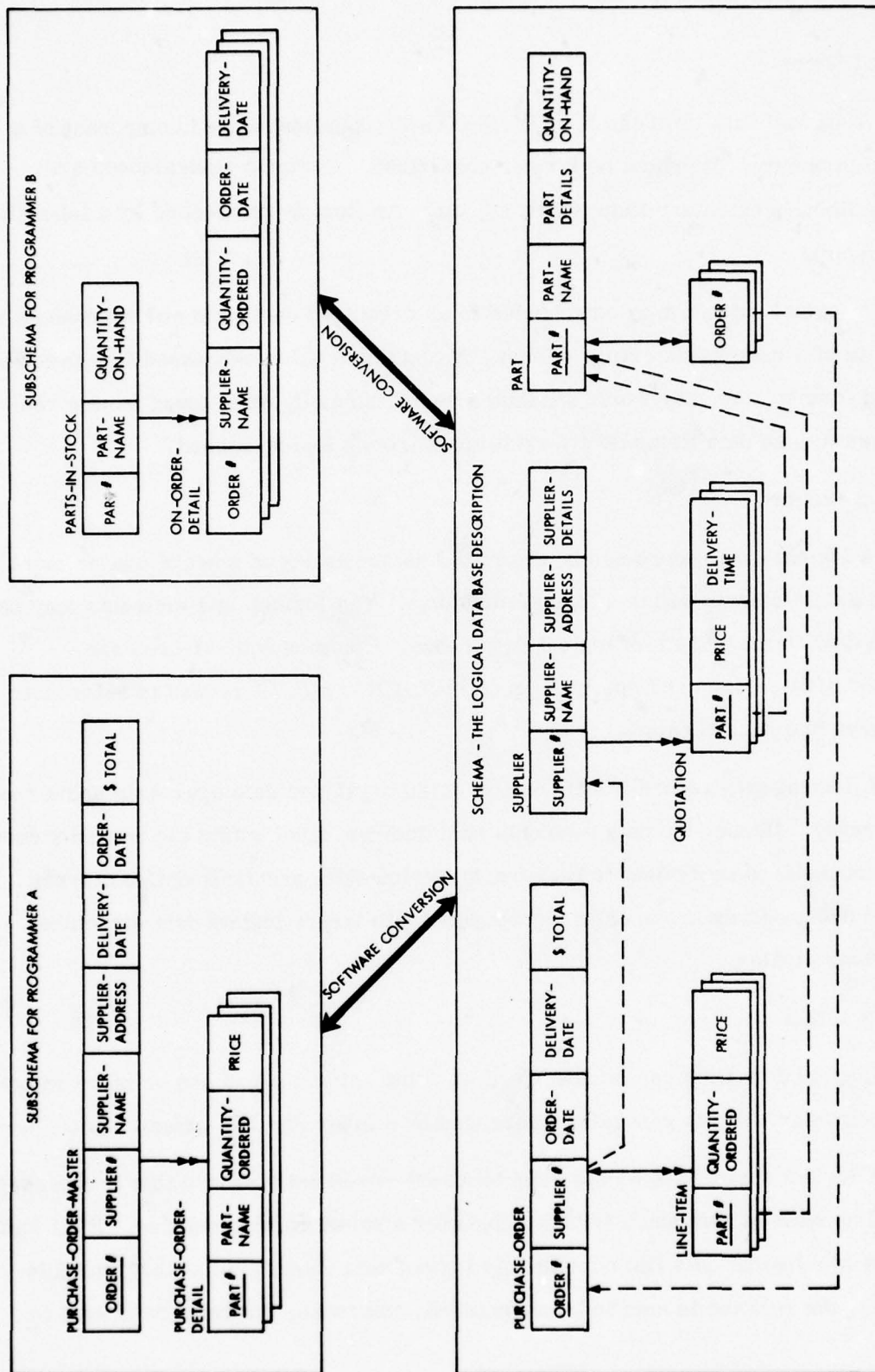


Figure 1-10. Overall Schema with Separate Subschemas

1.3.3.1 Item

A logical data item can be described as the smallest named component of a logical structure. It cannot be further subdivided. Common equivalences are: simple item, elementary item (COBOL), etc. An item is referenced by a label (the item name).

A logical data item is comparable to an organized data field and represents an attribute of a real-world entity. Hence, it contains a value expressed in a precisely defined format. As real-world attributes do not normally exist apart from a real-world entity, so data items do not exist apart from a logical record.

1.3.3.2 Record

A logical data record can be described as consisting of a set of one or more logical data elements within a logical structure. The logical data elements may be logical data items or sets of logical data items. Common equivalences are: segment (IBM), group and operating group (COBOL), etc. A record is referenced by a label (the record name).

A logical data record is comparable to an organized data aggregate and a real-world entity. Hence, the data elements on a common level within the record structure can be considered equivalent to real-world attributes of precisely defined format. Logical data records are usually incorporated into larger logical data structures called logical files.

1.3.3.3 Files

Logical data files can be described as comprising a set of one or more logical data records. A file is generally referenced by a label (the file name).

A logical data file is comparable to a real-world entity-set in that it represents a relation between common attribute values for a set of similar entities. If all logical records of a logical data file contain only logical data items (elementary items in COBOL), the relation is said to be normalized; otherwise, the relation is said to

be unnormalized. As discussed in Part III, the concept of normalized relations plays an important role in the development of relational data structures.

1.3.3.4 Base

The logical data base can be described as a collection of one or more logical data files. It is analogous to an organized data schema in that each logical data base is described by a specific schema. Within a model of an enterprise there may be many schemas; hence, there can be more than one logical data base. A logical data element may be included in different schemas; however, each usage is differentiated by virtue of the data element's relationship to other elements of the respective schemas. In any event, multiple use of logical data elements in the definition of organized schemas do not imply a logical intersection of said schemas. The placement of the actual data element is not entirely dictated by the schema, but depends on other factors such as space/time considerations.

1.3.4 Physical

The formatted structure (which is generally determined by SAs) can be described in terms of data, banks, extents, and blocks as listed under the "physical" column in Figure 1-7. Of course, the relationships between logical elements defined by the various data structures are considered in storing data in hardware; but frequency of use, available storage space, and other factors also influence the actual placement of data.

The correlation of physical storage terminology appearing in the literature with that appearing in Figure 1-7 is sometimes difficult. For instance, a file or data set is described in Engles [E] as a collection of extents. Sets were omitted here only to avoid confusion between logical and physical files. The user is again advised to look for possible variance in terminology when investigating new source materials.

1.3.4.1 Block

The block, or physical record, is a basic unit of data which is read or written as the result of a single input/output command by the computer hardware. It is the amount of data contained on a storage medium between inter-record gaps on tape or address markers on disk. Efficient utilization of the storage resources is a prime factor in determining the number of logical records contained within a physical record. In Figure 1-11, the block is the smallest unit described.

1.3.4.2 Extent

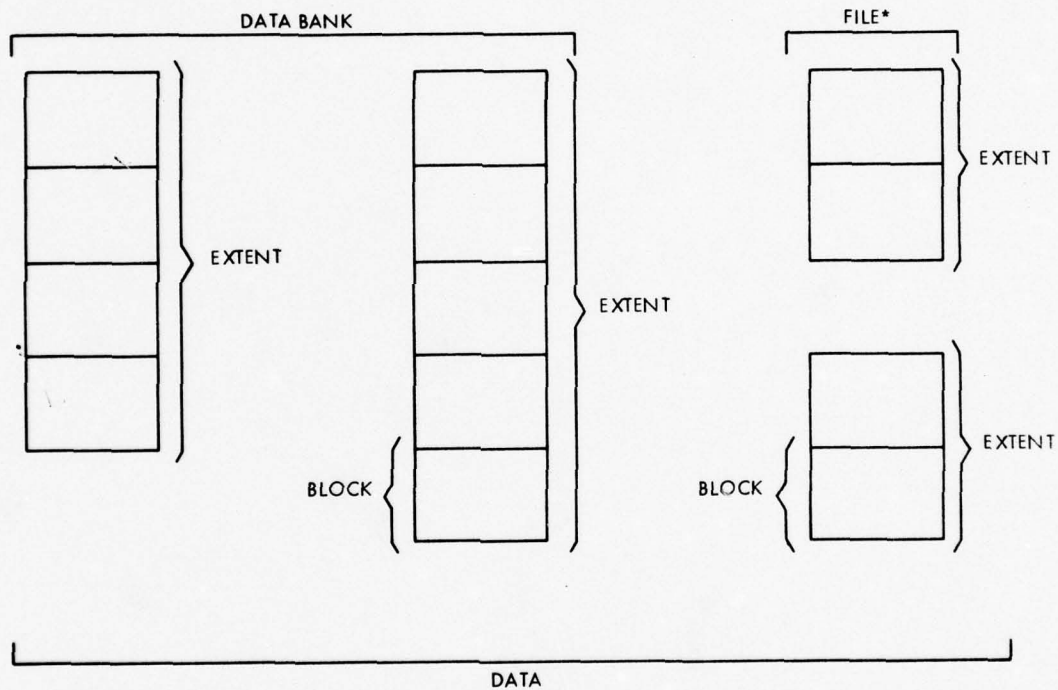
An extent is the contiguous region on an external storage medium for the placement of physical records. In most external storage devices, the extent is a multiple of the physical record size because of hardware restrictions in processing input/output requests from the device. The number of physical records contained within one extent will depend upon several factors, such as the type of storage media, the DBM supervisor, and the initial request for storage space. As requests for allocation/deallocation of space are processed, new extents will be linked/delinked providing logically continuous data, even though the information may be scattered across the media. However, a large number of extents may cause degradation of access time due to a high overhead in searching for a given record. As with the physical file, the logical file is defined, through the cataloging process, by one or more extents. The operating system then will provide the mapping function between the logical file known to the user and the hardware-related extents.

1.3.4.3 Bank

The physical data bank can be described as the totality of extents accessed by a DMS. The physical data bank is disjoint, i.e., the data in one bank is not related to data in another. To allow processing of data banks across computing systems, i.e., distributed data bank, a DMS may be global in nature containing functional capabilities on separate computer facilities.

1.3.4.4 Data

In the physical sense, data is the total information stored within a system, whether in a single computer or distributed over a network. Data encompasses all the banks and includes other stored information, such as operating system catalogs, directories, and temporary data. The term is used very broadly, but, when used in the physical sense, it is all-encompassing. When used in other contexts, i.e., data element, it is used as a modifier.



*IGNORING, FOR SIMPLICITY, THE SET (SEE FIGURE 1-7)

Figure 1-11. Relation of Physical Elements

CHAPTER 2 - HARDWARE

2.1 GENERAL

Hardware is defined as the physical equipment (devices and peripheral components) forming a computer system. The major hardware elements are: input/output (I/O) devices, central processing unit (CPU), and other storage devices.

Peripheral equipment includes auxiliary machines which may be placed under the control of the central computer, such as card readers, card punches, printers, and magnetic tape units. The devices under the control of the CPU are considered to be on-line equipment. Off-line equipment consists of those devices which are not in direct communication with the CPU.

2.2 DEFINITION OF TERMS

In defining terms related to computer hardware, it will be assumed that the reader is familiar with such traditional concepts as card reader, printer, and paper tape.

2.2.1 Central Processing Unit

The CPU is the heart of a computer system. It contains an arithmetic and logical section, a control section, and main memory (or core storage). The CPU performs arithmetic functions, makes logical decisions and controls the system components. This includes control of I/O devices and the storage of data into, or retrieval of data from, main storage (main memory). The CPU is sometimes referred to as the main frame.

2.2.2 Input/Output Devices

A computer system requires devices that enter data into and record data from the system. The oldest forms of input are punched cards and paper tape. Likewise, the oldest forms of output are punched cards, paper tape, and printed output. Magnetic tape, disc, and drum devices, although primarily used as storage devices, may also

be used for I/O. Other I/O devices include the keyboard terminal and cathode ray tube (CRT)* terminal. Input from a CRT may be via keyboard or light pen and output may be in the form of alphanumeric data or graphic displays. Terminals may be located at sites remote from the central computer site and information may be transmitted over telephone lines. Recent advances in technology have also produced an optical character recognition (OCR) device for input and a device for producing output in the form of microfilm.

2.2.2.1 Control Unit

The control unit coordinates input and output operations with the CPU. It directs the flow of information to and from the I/O devices. The control unit determines the priority for responding to requests for I/O service.

2.2.2.2 Channel

A channel manages the I/O control units and devices assigned to them. Once activated, a channel operates independently of the CPU, permitting I/O operations to be performed concurrently with CPU processing.

Figure 2-1 illustrates the relationship between the CPU, channel, control units, and I/O devices. A number of channels may be connected to one CPU.

2.2.2.3 Buffer

A CPU operates at a much higher speed than I/O devices. Since CPU operations and I/O operations can occur simultaneously, buffers are used to maximize utilization of the CPU. A buffer is a device used to temporarily store data prior to its transfer to core storage or prior to its output to some device. The transfer of data between a buffer and core storage is extremely fast in comparison with the transfer between the buffer and an I/O device. Therefore, while data is being stored in or retrieved from a buffer by an I/O device, the CPU can be performing other operations.

The overlapping of operations between I/O devices and the CPU is sometimes referred to as buffering.

*Honeywell terminology is Visual Information Processor (VIP).

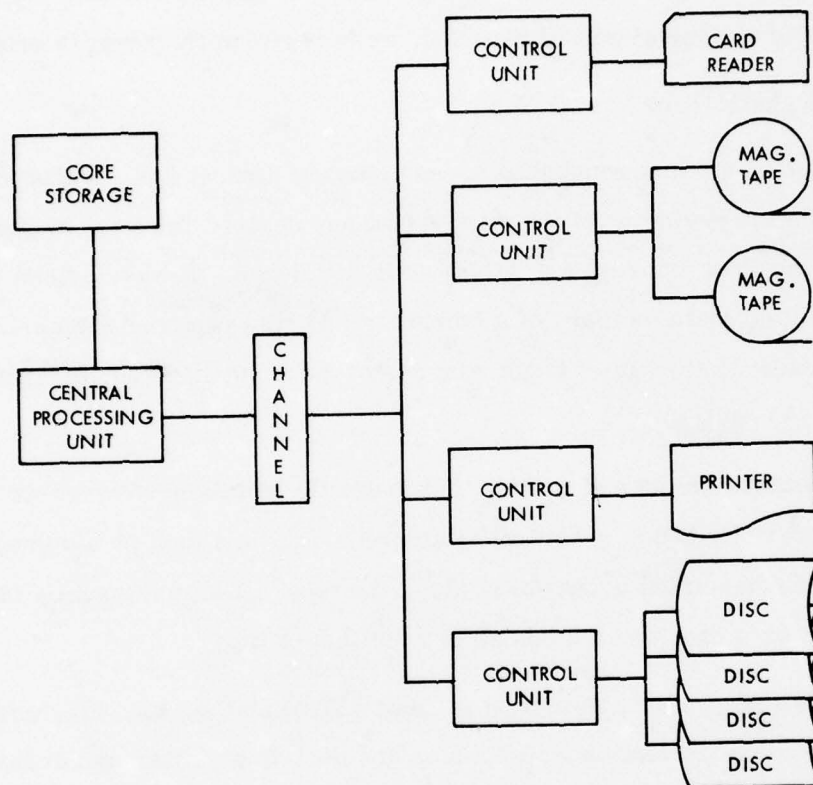


Figure 2-1. Input/Output Devices

2.2.3 Secondary Storage

Secondary storage is any storage device other than the main storage of a computer, such as magnetic tape, disc, or drum. Secondary storage usually holds much more information than main storage, but the access time is greater. Access time is the time required to find a specified storage location and transfer its contents to the arithmetic or logical unit of the CPU, or to perform the reverse process.

2.2.3.1 Magnetic Tape

Magnetic tape is a sequential access storage device; i.e., a tape reel must be read from the beginning of the tape to find any desired record. Magnetic tape was one of the first external storage devices developed. It was devised to supplement the limited main memory of a computer. It also replaced the earlier punched cards for external storage of large volumes of data. In addition, magnetic tape is used as an I/O device.

The actual reading and writing of a magnetic tape is performed as tape moves past the read/write head. Characters are recorded on a tape by a coding system consisting of magnetized areas (bits) along the tape. A tape is usually one-half inch wide, wound on a reel which holds up to 2400 feet of tape.

The writing of a magnetic tape is destructive; that is, any information previously written on the tape is written over and destroyed. Because of this, a protection device, a write ring, is used to prevent the accidental loss of information. The ring must be inserted into a mounted tape reel, if writing of the tape is to be permitted. Without a write ring, only reading of the tape is allowed.

2.2.3.1.1 Frame

A frame is a subdivision of a tape. It is a group of bits in one line across the tape, from side to side. One frame is similar to one column of a punched card.

The number of frames recorded per inch determines the density of a tape. Density can be measured in terms of characters per inch (CPI) or bytes per inch (BPI). A density of 800 BPI indicates that 800 bytes, or frames, are recorded on 1 inch of tape. Common tape densities are 200, 556, 800, and 1600 BPI.

2.2.3.1.2 Track

A track is one of several parallel paths along the length of a tape. It is similar to a horizontal row on a punched card.

A magnetic tape can contain seven or nine tracks. Both types are in use today. A nine-track tape contains eight tracks of data and one track for parity checking. Similarly, a seven-track tape contains six tracks of data and one parity bit per frame.

Parity checking is a technique based on an odd or even number of binary ones in a frame. In the representation of one character (eight or six tracks for a nine-track or seven-track tape, respectively), the parity bit is made either zero or one, whichever is required to make the number of ones in the frame an even number if the computer operates in even parity. Likewise, if the computer operates in odd parity, the parity bit is made a zero or one to make the number of bits in the frame an odd number. Parity bits are generated when a tape is written and the parity is checked when the tape is read. If the number of bits in any frame does not correspond to the parity of the computer, an error condition is indicated.

Figure 2-2 illustrates a nine-track tape of even parity.

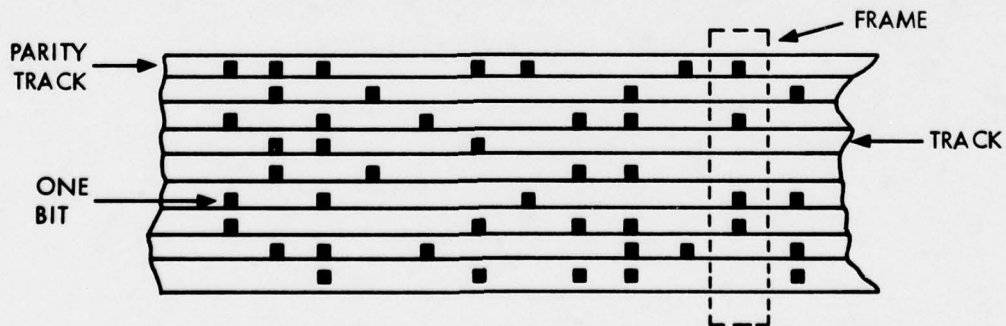


Figure 2-2. Nine-Track Tape

2.2.3.1.3 Block

On magnetic tape, information is stored in groups of characters. Each group is referred to as a block and may contain any number of logical records (See Paragraph 1.3.3.2). Blocks are separated by an interblock gap, simply a blank portion of tape.* The gap is produced each time a block is written. Upon reading, the information on the tape is read continuously until a gap is reached. The end of a file on tape is indicated by a tapemark, sometimes referred to as an end-of-file indicator.

Block lengths may be either fixed or variable. Many present day operating systems allow multi-file tapes, tapes which contain any number of files. Some systems also allow multi-reel files, files which extend over more than one reel of tape. The end of a reel of tape is indicated by a tapemark, an end-of-reel indicator.

Figure 2-3 illustrates a portion of a multi-file tape.

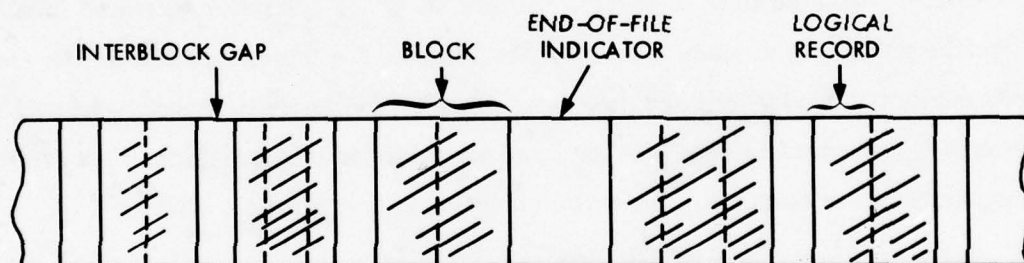


Figure 2-3. Multi-File Tape

*In some systems, the term "record" is used instead of block. When this is done, a "physical" record is implied.

2.2.3.2 Direct Access Storage Devices

A direct access storage device (DASD) is theoretically defined as a device in which the access times to all storage locations are equal. In other words, the time to obtain information from or place information into a particular location is independent of the location most recently accessed.

A DASD allows storage of extremely large quantities of data, in comparison with the main memory of a computer, but its access time is greater. The device may be on-line, directly accessible to the computer, or it may be removable, placed on-line when required.

A DASD is sometimes referred to as a random access storage device. A location may be accessed without reading the information preceding it on the device. This capability is in contrast to the sequential processing of magnetic tape. However, a DASD is capable of performing sequential, as well as random, processing.

2.2.3.2.1 Drum

The magnetic drum was one of the first DASD developed. A drum is a large cylinder that rotates at a constant speed. The outer surface is coated with a magnetic material for the recording and reading of data. Each magnetized spot represents a bit.

2.2.3.2.1.1 Track

Information is recorded in parallel tracks around a drum. Reading and writing are performed by read/write heads, which may be fixed or movable. A fixed head is stationary over a particular track; thus, each track has its own read/write head. Movable heads are attached to an arm which moves back and forth over a specified number of tracks, with access to only one track at a time.

A track may be one or more bits wide. By strict definition, a track is one bit wide and two or more tracks form a band.

2.2.3.2.1.2 Sector

Each track on a drum is divided into sectors. A sector is the smallest addressable unit of the drum and its size is usually predetermined by the device itself. When a sector is read, the content of the entire sector is transferred to main memory. When writing information onto a drum, one complete sector at a time is written.

Figure 2-4 illustrates a drum with fixed read/write heads (a) and with movable heads (b).

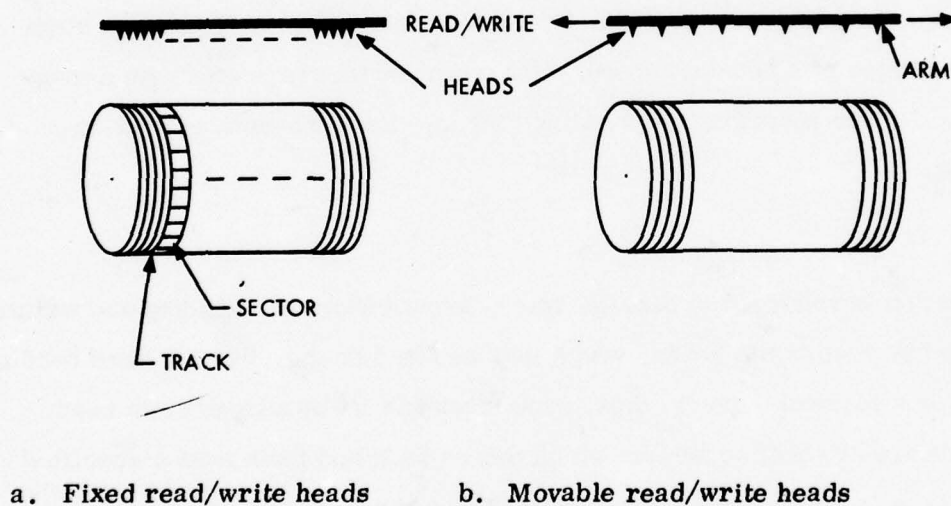


Figure 2-4. Drum

2.2.3.2.2 Disc

The most common DASD in use today is the magnetic disc, a thin plate coated on both sides with a magnetic material for recording data. One disc module contains a number of discs mounted on a common spindle, rotating at a constant speed. A disc unit may contain one or more modules. A disc pack consists of one module and may be either fixed or removable.

2.2.3.2.2.1 Track

A disc surface is composed of a number of concentric tracks on which data is recorded. Read/write heads perform the actual reading and writing. The heads may be fixed, with one head per track, or movable, where one head can access any one of a number of tracks. This concept is similar to the drum read/write heads (See Figure 2-4).

Disc read/write heads are attached to arms. One arm may contain two (or an even number) of heads, one for accessing the surface above the head and one for the surface below the head. See Figure 2-5(b).

2.2.3.2.2.2 Sector

The tracks of a disc are divided into sectors, where each sector contains an equal number of bits or bytes. * As with the drum, a sector is the smallest addressable unit of a disc. Sector size is usually determined by the device or the operating system of the computer.

2.2.3.2.2.3 Cylinder

A cylinder is the series of tracks, on a stack of discs, that are accessible without moving the read/write heads. These tracks form an imaginary cylinder.

Figure 2-5 (a) illustrates the surface of a disc. Figure 2-5 (b) illustrates movable read/write heads.

*Sector size is sometimes measured in terms of "words".

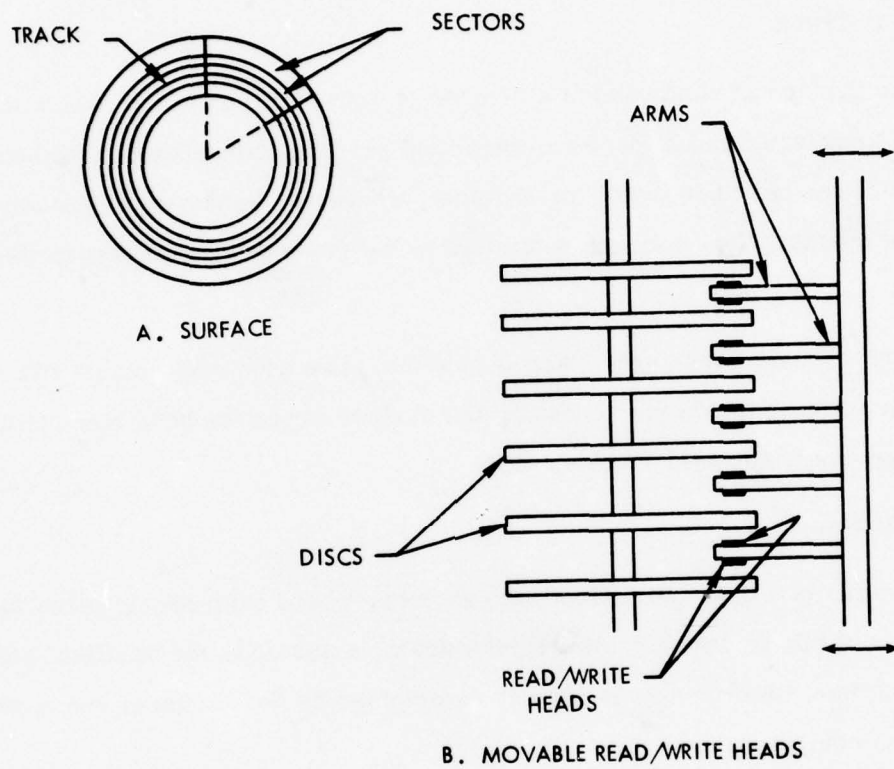


Figure 2-5. Disc

2.2.3.2.3 Data Cell

The data cell is another type of DASD, though not as commonly used as the disc and drum. A data cell device, or drive, consists of a number of cells which are divided into subcells. Each subcell contains magnetic strips on which information is recorded in tracks across each strip.

To access a strip, it is selected from the appropriate subcell and wrapped around a small rotating drum above the cells. A set of read/write heads is attached to the drum to perform the reading and writing as the strip moves past the heads. When processing of one strip is completed, it is returned to its subcell and the next desired strip is selected.

A data cell device is illustrated in Figure 2-6.

2.2.3.3 Terabit Memory System

The Terabit Memory (TBM*) System is one of the most recent developments in storage devices. The TBM is considered to be a Mass Storage System (MSS), an extension of current DASD. It provides storage of a trillion bits, or 125 billion bytes, and increases the capacity for on-line data bases by several orders of magnitude over DASD. TBM appears to the user to be a virtual disc.

The storage medium of the TBM system is 2-inch wide magnetic tape, as used in commercial color television recorders. Data is recorded across the tape, rather than longitudinally as on standard magnetic tape storage devices. The data is written in fixed length blocks (physical records) and each block is identified by a unique address.

When a job requires data that resides on a TBM tape, the system transfers that data to a DASD. The transfer is transparent to the user; he simply requests a collection (set) of data by a unique name with no concern as to the physical location

*Ampex trademark

of the data. A job begins execution when all of its required data has been transferred to a DASD. Upon job completion, data sets which have been modified are transferred from the DASD to the TBM tapes. Figure 2-7 illustrates a TBM system serving several host CPUs.

The TBM System is a great deal more than tape storage. It is a system composed of a memory section and a control section.

In Figure 2-7, staging controllers are placed between the DASDs and the TBM tapes to coordinate the transfer (staging) of data. In addition, a resource controller exists for communication between the host CPUs and the staging controllers.

2.2.3.3.1 Memory Section

The memory section of the TBM System consists of transports, transport drivers, and data channel units. A transport module contains two transports, each with one TBM tape. Transports provide the mechanism for moving the tapes. The transport driver contains a small computer and directs all activities of the transports. The data channel unit contains the electronics to perform simultaneous read/write operations.

2.2.3.3.2 Control Section

The control section of the TBM System consists of interface buffers, channel control hardware and a special purpose computer, the System Control Processor (SCP).

A data request initiated by a host CPU is interpreted by the SCP. The SCP allocates the appropriate resources and the data is retrieved from the memory section of the TBM. The data is transferred to an interface core buffer and made available to the host. For storing data, the SCP dedicates the required interface buffers, enabling the transfer of data from the host CPU.

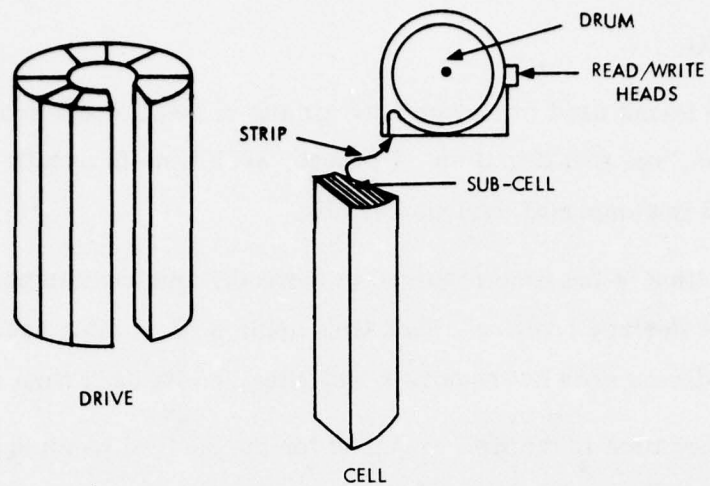


Figure 2-6. Data Cell Device

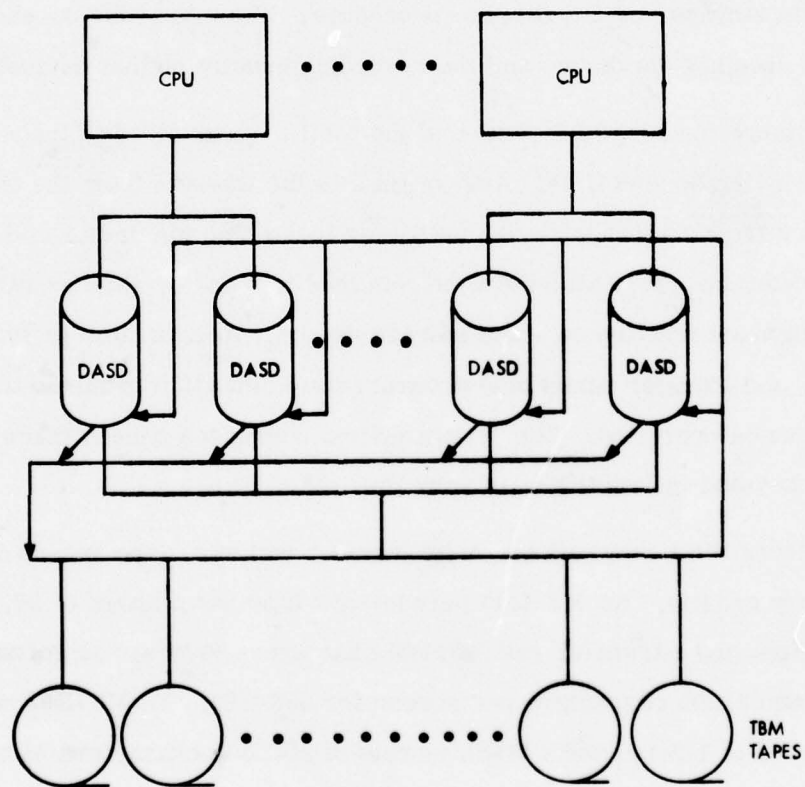


Figure 2-7. Terabit Memory System

2.3 TIMING

Some terms used to measure the timing of DASD operations are: seek time, latency time, and transfer time. Of these, seek time is usually predominant and can be used in comparing various devices.

Seek time is the time required to move the arm containing the read/write heads to the desired position. Seek time applies to movable head devices only. A fixed head device does not require positioning, so its seek time is zero.

Latency time is the time required for the desired location of a device to come under the read/write head. Latency time, sometimes referred to as rotational delay, is determined by the rotation speed of the device.

Transfer time is the time required to read a record on a device and transfer it to main storage, or the reverse procedure. Transfer time depends upon the rotation speed of the device and the recording density on that device.

A more meaningful measure of the total time involved in transferring data is given by the access time. Access time is the interval from the time data is requested from a storage device until it is located on the device and transferred to main storage. It is also the time required to transfer data from main storage to a designated location on some storage device. Access time includes the seek, latency, and transfer times plus program time, the time required to initiate the read or write operation. The program time, being the time it takes the CPU to execute several instructions, is very low.

Among the Honeywell magnetic tape subsystems, tape speeds and transfer rates vary greatly. An MTH200 seven-track tape has a speed of 37.5 inches per second (ips) and a transfer rate of 7500 characters (6 bits) per second for 200 BPI density and 21000 characters per second for 556 BPI. An MTH505 nine-track tape has a speed of 125 ips and a transfer rate of 267,000 characters (6 bits) or 200,000 bytes (8 bits) per second for a 1600 BPI tape.

Some typical timings for access to disc storage are given in reference to Honeywell equipment. The DSS181 subsystem provides a transfer rate of 416,000 characters or 312,000 bytes per second. The average seek time for the DSS181 is 34 milliseconds and average latency time is 12.5 milliseconds. The DSS190 subsystem provides faster access. Its transfer rate is 1,074,000 characters or 806,000 bytes per second. The average seek time for the DSS190 is 30 milliseconds and the average latency time is 8.3 milliseconds.

In comparison to disc storage, a magnetic drum offers lower access times. A data cell, in contrast, is a much slower device, with access times higher than disc storage.

For the TBM System, access time ranges from 1 to 30 seconds, much higher than any DASD.

2.4 CAPACITY

Capacity is the amount of information that can be stored on a device. Capacity is usually measured in terms of characters or bytes, but may also be in terms of words. Caution must be used when dealing with measurement by word count, since word size varies from one computer to another.

The capacity of magnetic tape depends upon the density at which information is recorded. A high density tape (1600 BPI) can contain more characters than a low density tape (200 BPI). Block size must also be considered, since interblock gaps (approximately 0.5 to 0.75 inches each) appear between blocks.

A magnetic drum is considered to be a low capacity DASD. It can usually accommodate about four million characters.

The capacity of a disc unit depends upon the number of discs in a module and the number of modules in a unit. For example, a Honeywell DSS181 on-line disc unit with three spindles has a capacity of 82 million characters or 62 million bytes. With 16 spindles, the DSS181 can accommodate 442 million characters or 331 million bytes. The DSS190 disc unit has a capacity of 266 million characters or

200 million bytes on a unit with two spindles and 2.13 billion characters or 1.6 billion bytes for 16 spindles. Removable disc packs range in capacity from 27,648,000 characters or 20,736,000 bytes to 133 million characters or 100 million bytes for the DSS181 and 190 subsystems, respectively.

The data cell is considered to be a large capacity storage device. One data cell drive can generally accommodate over 400 million characters.

The TBM System provides an extremely large storage capacity. One TBM tape can hold the equivalent of 2500 average packed or 500 fully packed 800 BPI tapes. This is equivalent to 55 fully packed IBM 3330 disc packs. In general, capacities of magnetic tape and DASD's range from 10^7 to 10^{10} bits, while the TBM capacity ranges from 10^{11} to 10^{12} bits.

2.5 COST

Generally, the high speed storage devices (i.e., low access time) are more expensive than low speed devices. Main storage (core storage) is the most expensive means of storing data. Magnetic tape is least expensive of all external storage devices and costs vary considerably among the direct access storage devices. The Terabit Memory System offers the least expensive means of storing data on-line.

The most meaningful measure of cost, other than the storage medium itself, is the cost per bit of storage. For a 1600 BPI tape, the cost is approximately 5×10^{-5} cents/bit for an average packed tape (not fully packed). The cost for disc storage is approximately 9×10^{-5} cents/bit. For TBM storage, the cost is reduced to 5×10^{-7} cents/bit, approximately. Average tape costs are 10 dollars for a standard magnetic tape and 150 dollars for a TBM tape. The cost of a disc pack ranges from about 250 to about 750 dollars, depending on the model.

2.6 DEVICE EVALUATION

Access time, capacity, cost, and file structure* should be considered in selecting a storage device. Magnetic tape, because of its low cost, may be used for

*File structures and access methods are discussed in Chapters 3 through 7.

sequential files that do not require frequent access or update. Tape is most often used to store backup copies of files residing on a DASD (See Paragraph 10.3.1).

A sequential file may be stored on tape or DASD. A random access file must be stored on a DASD. Among the devices available, the magnetic drum offers fast access, but the capacity of a drum is limited. Disc storage offers a larger capacity with higher access time. A desirable feature of disc storage is the removable disc pack for data that need not be on-line at all times. The data cell provides a much larger capacity than disc storage; however, access time for data cell storage also increases considerably. The disc is the most popular DASD in use today.

The TBM System provides on-line storage several orders of magnitude larger than any DASD. However, TBM access time is much higher than DASD's. The most practical use of a TBM System is seen to be as a replacement for tape and disc libraries. Thus, all required data is always on-line and human intervention, as mounting tapes and disc packs, is eliminated. The relatively low cost of TBM storage adds to the desirability of this system for storing extremely large volumes of data on-line. To date, however, the TBM System has been interfaced with only IBM 360 and 370 CPUs.

PRECEDING PAGE BLANK - NOT FILMED

PART I

Part I discusses serial structures including methods, techniques and other topics directly related to these structures.

Other methods and techniques, utilized to serialize data stored in random data structures and access data sequentially, will be found in Part II.

CHAPTER 3 - SERIAL STRUCTURES

3.1 GENERAL

A serial structure is defined as a set of logical data elements* whose organizational properties essentially involve only linear (one-dimensional) relationships. Addressing elements within a serial structure is a function of the specific set of ordering rules used for both placement and access. Placement relations may be established through:

1. Physical ordering
2. Logical ordering.

3.2 PHYSICAL ORDERING

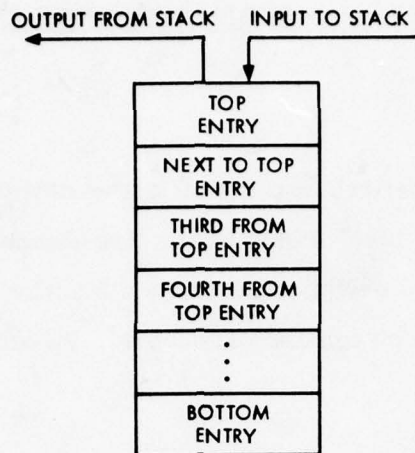
Physical ordering relates each logical element through physical juxtaposition to the next logical element in the sequence. Physical ordering can be accomplished through application of:

1. Algorithmic techniques, such as stacks, queues and dequeues. (See Figure 3-1).
2. Sorting techniques applied against a defined key field imbedded in the data elements (See Figure 3-2.)

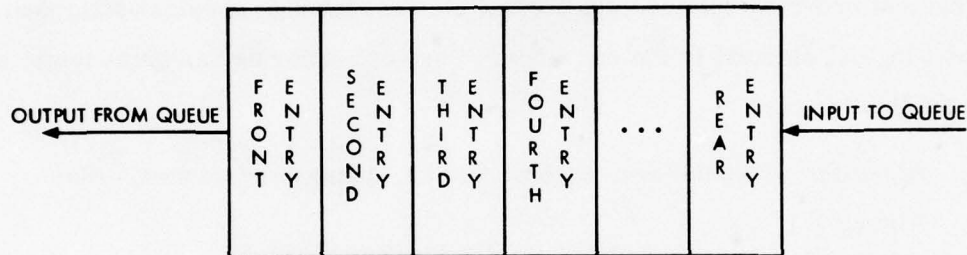
Figure 3-1 illustrates physical ordering of five different serial structures using algorithmic techniques. Figure 3-1 (a) and (b) depict a physically ordered stack, queue, and various deque** configurations, respectively. As indicated in the figure, a stack is usually referenced from the "top", and represents a typical last-in-first-out (LIFO) list. A queue is usually referenced from the "front" or from the "rear", and represents a typical first-in-first-out (FIFO) list. A deque is usually referenced from the "leftmost" or "rightmost" element, and is the most general algorithmic form. By applying appropriate restrictions on the input and/or output of a deque, it can be made to behave as a stack or as a queue. If unrestricted, a deque can be used to describe

*Logical data element and logical record are synonymous terms as used here.

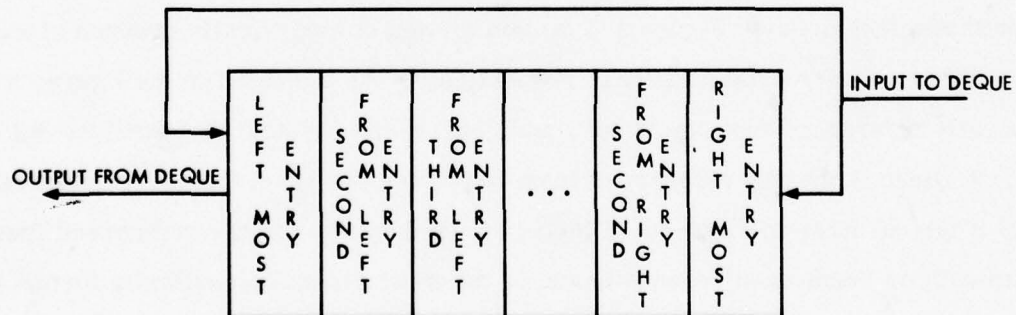
**Deque is pronounced as deck.



(A) PHYSICALLY ORDERED STACK

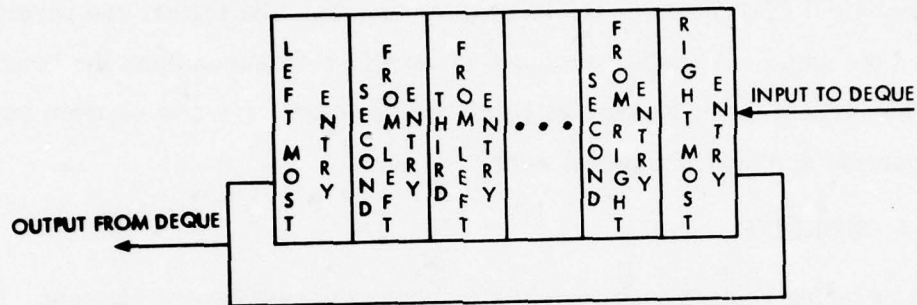


(B) PHYSICALLY ORDERED QUEUE (EQUIVALENT TO A PHYSICALLY ORDERED INPUT AND OUTPUT RESTRICTED DEQUE).

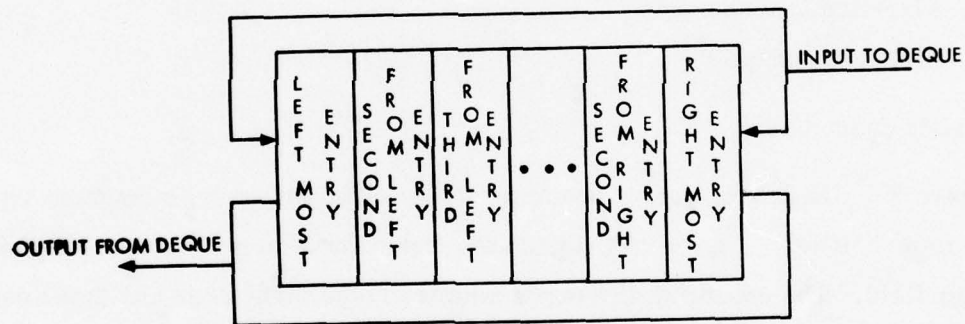


(C) PHYSICALLY ORDERED OUTPUT RESTRICTED DEQUE.

Figure 3-1. Serial Structures Physically Ordered Through Application of Algorithmic Techniques (1 of 2)



(D) PHYSICALLY ORDERED INPUT RESTRICTED DEQUE.



(E) PHYSICAL ORDERED DEQUE (NOT RESTRICTED).

Figure 3-1. Serial Structures Physically Ordered Through Application of Algorithmic Techniques (2 of 2)

complicated ordering algorithms. Unrestricted deques generally are of more interest to the theoretician than to the practitioner. In the figures, the terms input and output indicate insertion and deletion of logical elements.

Figure 3-2 illustrates physical ordering through sorting on a defined key field. The defined key-field EMP contains the employee number. The logical and physical structures are the same. A sorted structure is usually referenced from the "current" element.* The current elements may be retrieved, deleted, or a new element inserted between the current and next logical element.

3.3 LOGICAL ORDERING

Logical ordering relates each logical element to the next logical element. It is generally accomplished through the use of pointers appended to data elements to form a linked list, rather than through physical positioning. Manipulation of the pointers can be accomplished through application of:

1. Algorithmic techniques
2. Sorting techniques

as previously described.

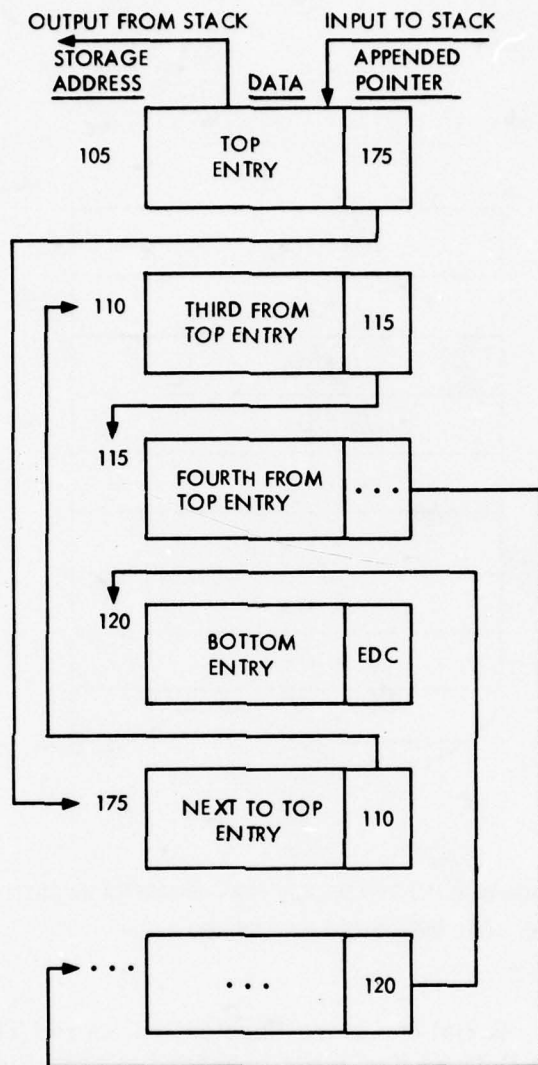
Figure 3-3 illustrates logical ordering of five different serial structures using algorithmic techniques. Figure 3-4 illustrates logical ordering through sorting on a defined key field. The essential difference between these structures and those depicted in Figures 3-1 and 3-2 lies solely in the method selected for establishing and maintaining logical relationships between the structures. In all other respects, the structures are equivalent.

*The "current" element is the first element in the structure where an operation begins.

EMP # 115
EMP # 120
EMP # 130
EMP # 175
EMP # 210
• • •
• • •
• • •
• • •

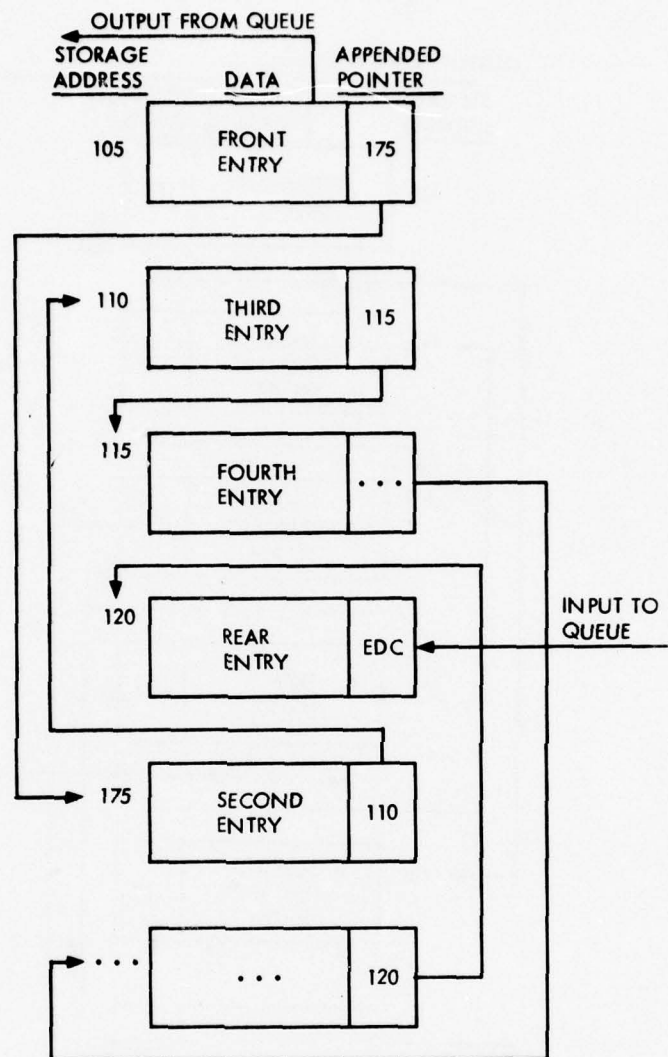
- EMP (EMPLOYEE NUMBER) IS LOGICAL KEY FIELD IMBEDDED IN EACH DATA ELEMENT
- LOGICAL AND PHYSICAL SEQUENCES ARE EQUIVALENT.

Figure 3-2. Serial Structure Physically Ordered Through Sorting on a Defined Key Field Imbedded in Data Elements



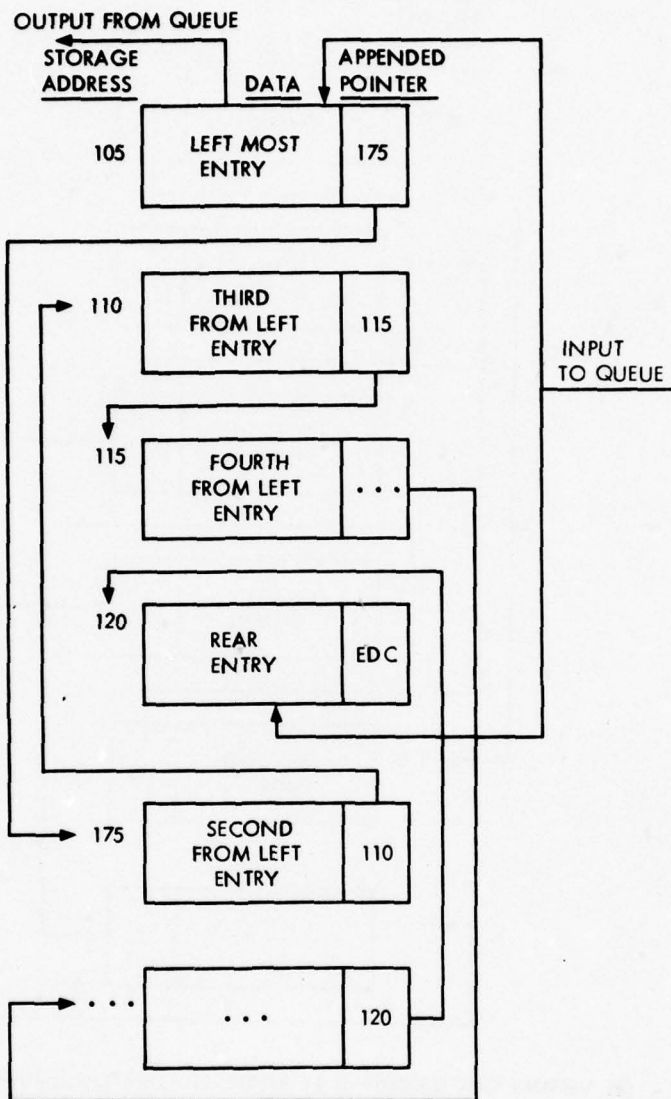
(A) A VIRTUAL STACK ORDERED BY APPENDED POINTERS.

Figure 3-3. Serial Structures Logically Ordered Through Application of Algorithmic Techniques (1 of 5)



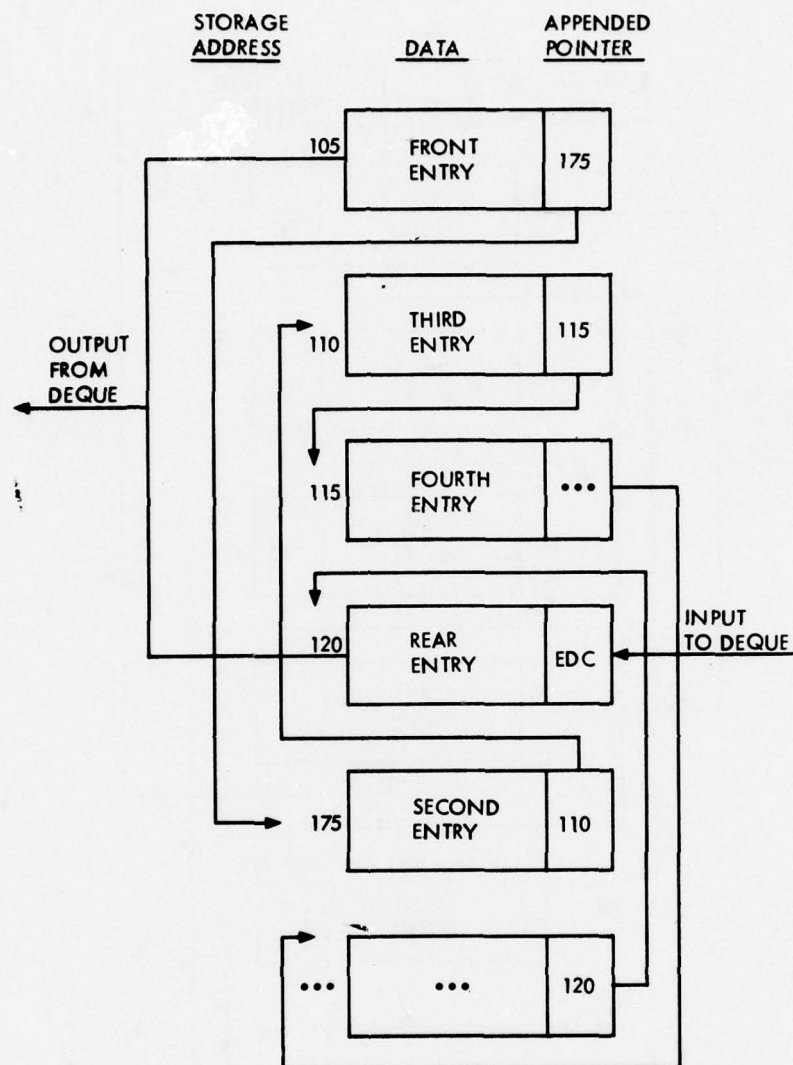
(B) VIRTUAL QUEUE ORDERED BY APPENDED POINTERS (EQUIVALENT TO A VIRTUAL INPUT AND OUTPUT RESTRICTED DEQUE ORDERED BY APPENDED POINTERS).

Figure 3-3. Serial Structures Logically Ordered Through Application of Algorithmic Techniques (2 of 5)



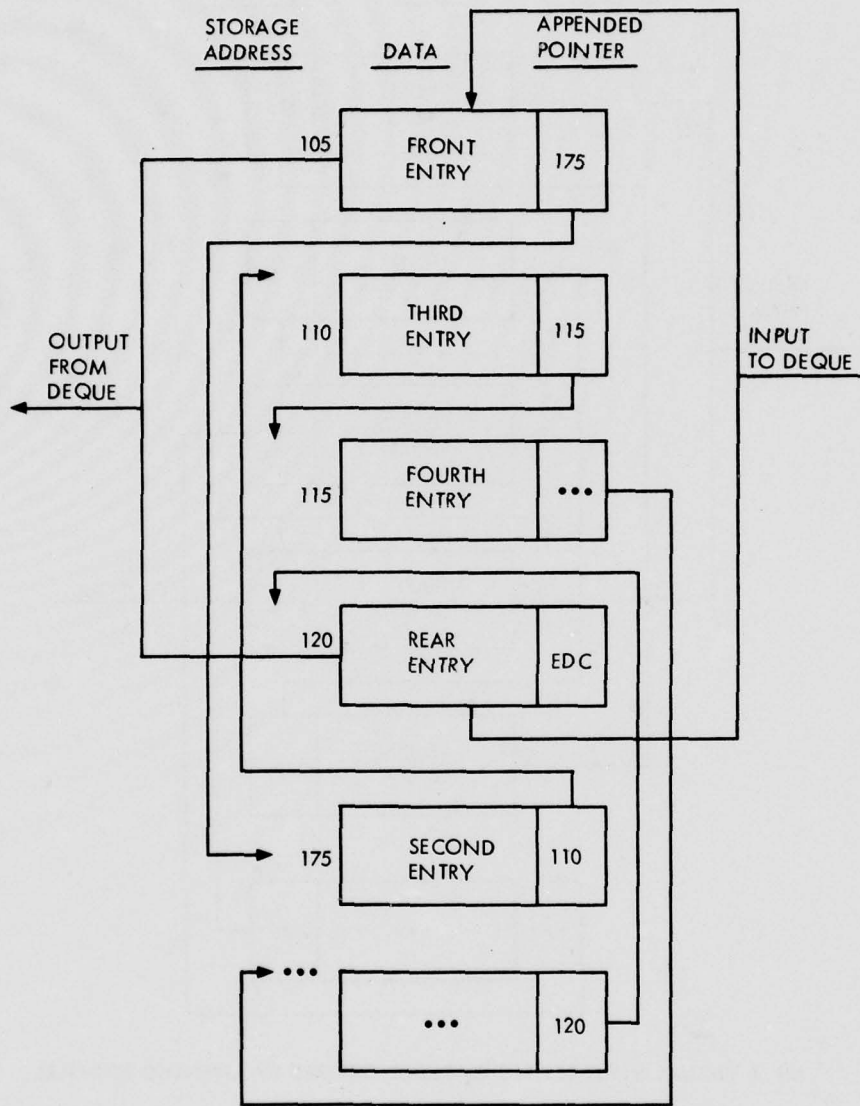
(C) A VIRTUAL OUTPUT RESTRICTED DEQUE ORDERED BY APPENDED POINTERS.

Figure 3-3. Serial Structures Logically Ordered Through Application of Algorithmic Techniques (3 of 5)



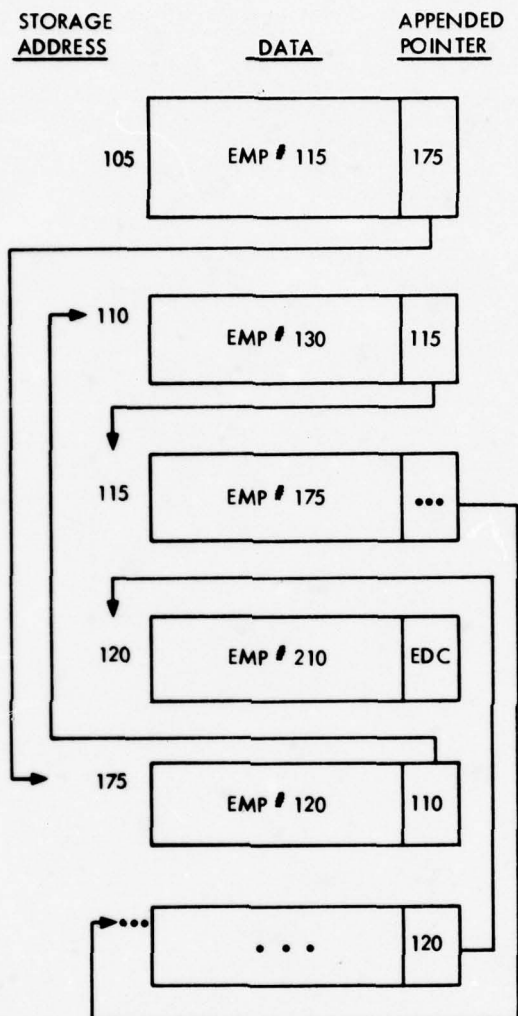
(D) A VIRTUAL INPUT RESTRICTED DEQUE ORDERED BY APPENDED POINTERS.

Figure 3-3. Serial Structures Logically Ordered Through Application of Algorithmic Techniques (4 of 5)



(E) A VIRTUAL DEQUE ORDERED BY APPENDED POINTERS.

Figure 3-3. Serial Structures Logically Ordered Through Application of Algorithmic Techniques (5 of 5)



- EMP (EMPLOYEE NUMBER) IS LOGICAL KEY FIELD IMBEDDED IN EACH DATA ELEMENT
- LOGICAL AND PHYSICAL SEQUENCES ARE NOT EQUIVALENT
- EDC DENOTES END-OF-CHAIN.

Figure 3-4. Serial Structure Logically Ordered Through Sorting on a Defined Key Field Imbedded in Data Elements Through Appended Pointers

CHAPTER 4 - SEQUENTIAL ACCESS OF SERIAL STRUCTURES

4.1 GENERAL

Serial structures are accessed through a current data element which may be the first or any other element in the structure. Sequential access methods insert after the current logical data element, delete the current logical data element, and retrieve the next logical data element. If there is no next element, an end-of-data condition is indicated. Even though a defined key field exists, it cannot be utilized directly to indicate a desired logical data element, but, instead, a user must serially traverse the logical element sequence until the desired element is located.

4.1.1 Insertions/Deletions

If logical relationships are maintained through physical positioning, as illustrated in Figures 3-1 and 3-2, every insertion and deletion will require physical movement of the data within the data structure to express the new logical relationships. If logical relationships are maintained through pointer linkage, only adjustments in affected pointers are required to express the new logical relationships. The insertions themselves can always be stored in the next available data space location.

4.1.2 Flagging

Flagging is used to delete logical data elements from a data structure. It is a logical operation and does not require repositioning of data or modification of pointer linkage. The data space occupied by flagged elements cannot be reused until the next reorganization (rewrite) of the data structure.

4.1.3 End-of-Space Condition

Generally, one or more physical data space descriptors are associated with every data structure. When the data space for any serial structure is exhausted, an end-of-space condition is indicated. Insertions can not be made until additional free space is obtained. Free space may be obtained by allocating the space formerly used by flagged elements and freed during reorganization of the data structure or by extending the current space allocation.

4.1.4 Element Formats

Sequential access methods generally apply to both fixed and variable length formats. Variable length formats are defined by appending a count control to the logical data elements.

4.2 ACCESS METHODOLOGY

Access methods applicable to serial structures can be classified as serial or sequential.

4.2.1 Serial Methods

Serial techniques can be used for stacks, queues, and deques.

4.2.1.1 Stacks

A stack can be defined as a linear list of elements where all insertions and deletions are usually made at the top. Therefore, access mechanisms operating on stacks utilize:

1. Top of the stack pointer
2. Data space delimiters,

and provide for:

- Initialization of stack controls
- Serial accessing of stack elements
- Counting of the number of logical elements currently in the stack.

Figure 4-1 illustrates a simple mechanism which fills the data space from the bottom up and empties it from the top. The net result is a last-in-first-out (LIFO) list processor.

4.2.1.2 Queues

A queue can be defined as a linear list of elements where all insertions are made at one end of the list, and all other accesses are made at the other end. Therefore, access mechanisms operating on queues utilize:

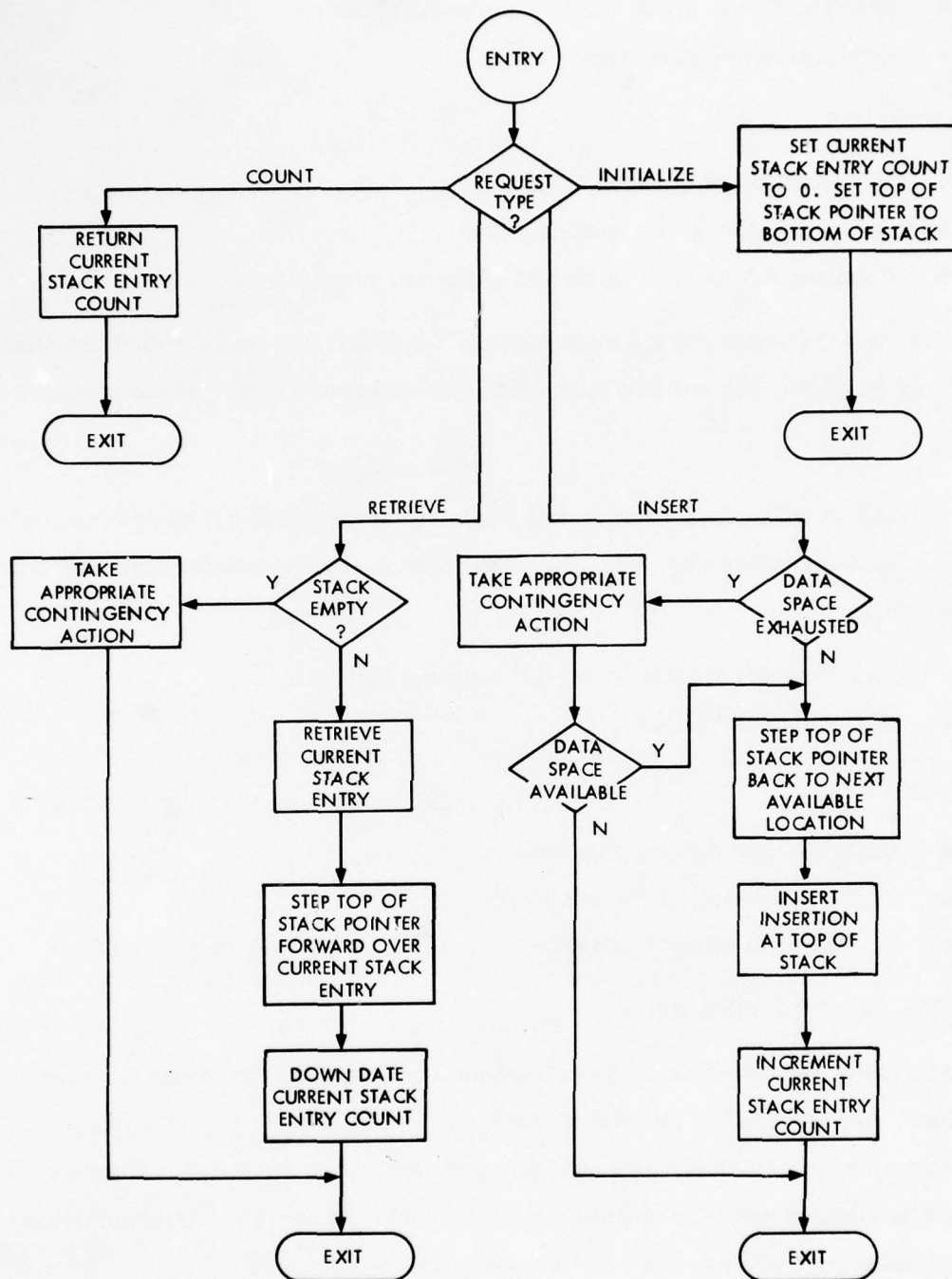


Figure 4-1. Flow Diagram of a Sequential Access Mechanism Designed to Operate on Physically Ordered Stacks

- Front and rear queue (logical element) pointers
- Data space delimiters,

and provide for:

- Initialization of queue controls
- Serial accessing of queue elements
- Counting the number of logical elements currently in the queue.

Figure 4-2 illustrates a simple mechanism which fills and empties the queue from front to rear. The net result is a first-in-first-out (FIFO) list processor.

4.2.1.3 Deques

A deque is defined as a linear list of elements for which all insertions, deletions, etc., are made at either end of the list. Therefore, access mechanisms operating on a deque utilize:

- Leftmost and rightmost logical element pointers
- Data space delimiters,

and provide for:

- Initialization of deque controls
- Serial accessing of deque elements
- Counting the number of logical data elements currently in the deque.

4.2.1.3.1 Restricted Deques

An output restricted deque is defined as one in which all accesses, except insertions, are confined to one end of the deque. An input restricted deque is defined as a deque in which all insertions take place at one end of the deque. Finally, an output and input restricted deque is defined as one in which all accesses are restricted as specified separately for input and output restricted deques.

Depending upon the nature of the restrictions imposed, an input and output restricted deque can take on the characteristics of a stack or a queue. This is illustrated in Figure 4-3.

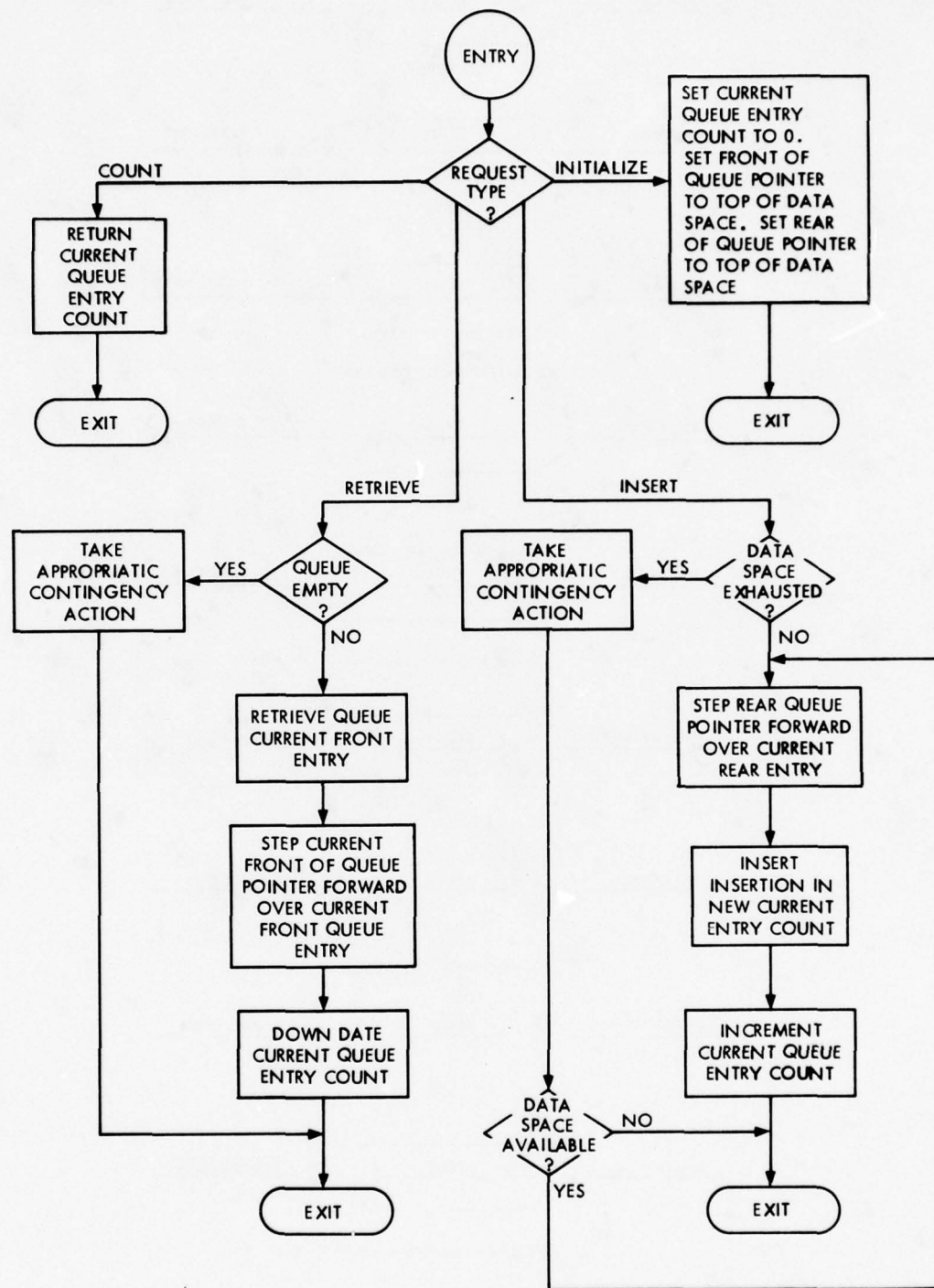
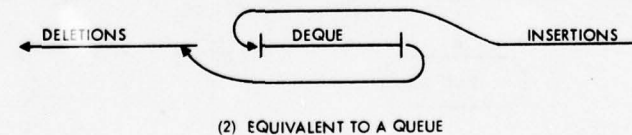
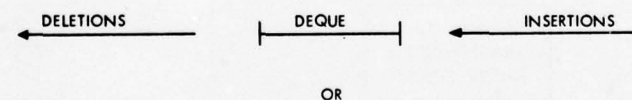
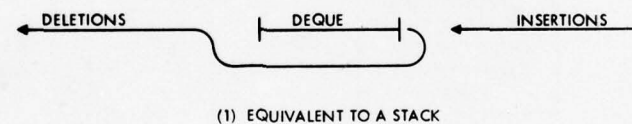
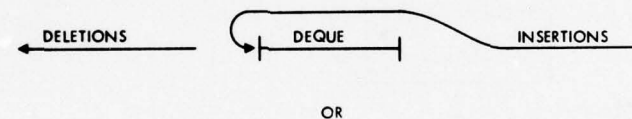
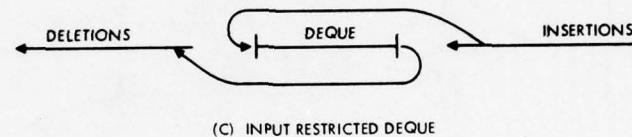
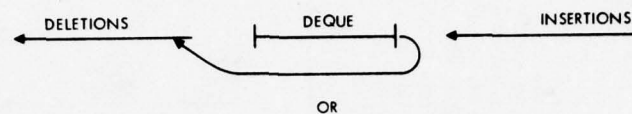
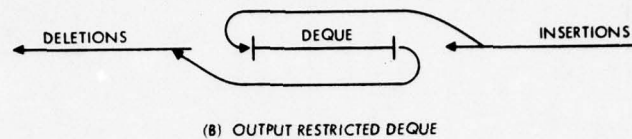
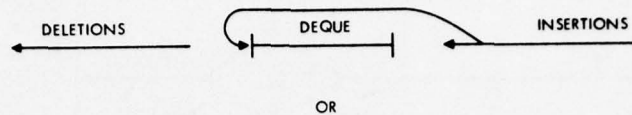
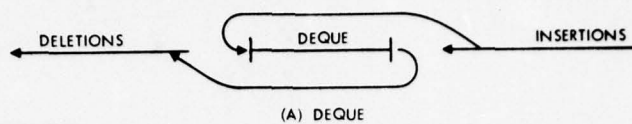


Figure 4-2. Flow Diagram of a Sequential Access Mechanism Designed to Operate on Physically Ordered Queues



(D) OUTPUT AND INPUT RESTRICTED QUEUE

Figure 4-3. Data Flow When Serially Accessing Serial Data Structures Using Algorithmic Techniques

4.2.2 Sequential Methods

Sequential methods operate on linear lists of logical data elements in which access is a function of the collating sequence applied against a defined key field imbedded in the data elements. Even though the structure is dependent upon the value of the imbedded key, sequential access methods do not provide the user with the ability to selectively access logical data elements by virtue of key value, but obligate him to sequentially traverse the serial structure, beginning at the first logical element, until the desired logical element is located.

4.2.2.1 Data/Device Independence

The linked structures illustrated in Figures 3-3 and 3-4 display a measure of data independence in that the physical representation in storage is relatively independent of the logical structure (user's view). By replacing hardware address pointers appended to logical data elements (and physical records, if blocked), with pointers computed relative to a common reference, the data will also take on a measure of device independence.

4.3 ACCESS CONSIDERATIONS

In general, techniques associated with serial scanning of data structures lend themselves to sequential batch processing applications. Algorithmic techniques are uniquely suitable for a variety of specialized scientific and language translation applications.

Physically sequenced structures are most efficient when undergoing periodic batch updating and/or reporting applications that effect a substantial proportion of elements in the data structure.

Logically sequenced structures are most efficient when supporting highly volatile data environments; however, the progressive degradation in processing efficiency caused by the dispersion over a broad spectrum of the data space, as a result of frequent update activity, may necessitate frequent reorganization of the data structure. Logical sequencing can also be used where logical data elements are allocated

dynamically, out of a common buffer pool that is shared by more than one data structure.

Other uses for which sequential access methods are generally preferred include archival data storage, storage of infrequently processed files, and very short files.

Algorithmic techniques such as stacks, queues and deques are preferred in connection with recursive user algorithms. Applications include parsing of computer language, evaluation of arithmetic and boolean expressions, ALGOL procedure linkage, and high precision algebraic operations (Knuth [H, Vol. 2]).

4.4 ADVANTAGES

The principal advantages of sequential access of serial structures are:

- Physically ordered structures are readily accessed via sequential access mechanisms and can be stored on any medium, including low cost devices
- Serial structures stored on card and tape devices are always stored in an optimized state insuring the most efficient access
- Sequential access capability is available through practically all computer data management systems
- Serial structures and sequential methods are easy to understand and use.

4.5 DISADVANTAGES

The principal disadvantages associated with sequential access of serial structures are:

- Updating a physically ordered data structure requires physical movement of data within the structure which, depending upon the storage media, may require a complete rewrite of the entire structure.
- Addressing a particular logical element of the structure requires a serial scan of the structure beginning with the first logical element until the desired element is located

- Applying updates efficiently against a serial structure, where logical relationships are a function of a defined key field, requires that the updates be in the sort order of the defined key.

4.6 VARIATIONS

To make the most efficient use of computing resources, several techniques such as:

- Blocking
- File Limits

have been developed in connection with operations on serial data structure.

4.6.1 Blocking

Blocking is a technique frequently used to efficiently utilize storage space and minimize the time required to access data structures stored on-line on devices such as disks, drums, tapes, etc. The block size is generally a function of such storage unit characteristics as sectors, track length, etc. The number of logical data elements per block (physical record) is referred to as the blocking factor. One or more logical data elements are generally "blocked" into a physical record; a logical data element exceeding the maximum block size may occupy one or more physical records. To minimize accessing overhead, blocks occupied by oversize logical data elements are not used for other data elements. Appropriate truncation of the previous block and last block occupied by the oversize element can be accomplished either through the block control parameters, "padding" of unused space, or use of an "end-of-block" flag.

"Deblocking" is the inverse operation of "blocking".

It is customary to describe block content by using a block control appended to the physical record. The block control generally specifies the length of the block or number of logical elements contained therein, and sets a flag which indicates whether the block is or is not a continuation of the previous block.

Blocked records may be physically or logically ordered on a device, irrespective of the logical data structure order. Hence, the physical structure may vary somewhat from the logical structure, but such variance is "transparent" to a user.

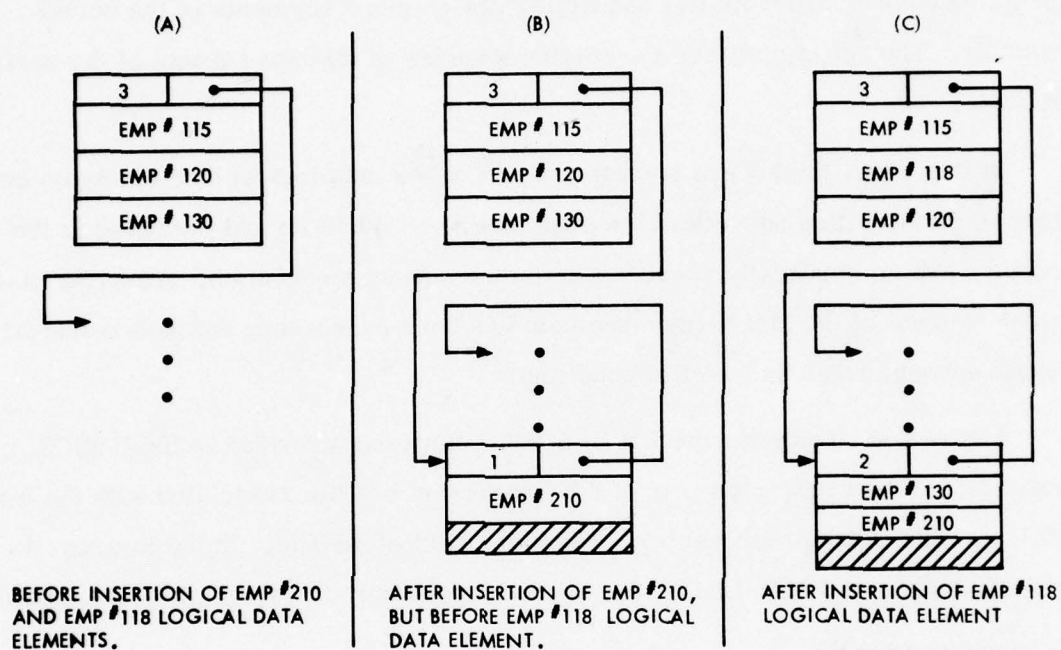
When physical records are physically ordered, insertion of additional logical data elements always requires movement (rewrite) of data elements at least from the point of insertion.

If physical records are logically ordered on a storage device, the record control will include a pointer to the next record, or, if none exists, will indicate an end-of data condition. Insertions may be stored in those existing physical records which have enough free space to wholly contain them. If not, the physical records are "split",* and the existing logical data elements are moved to new blocks to provide space for the new logical data elements. Splitting is accomplished so that a record is accessed once, per transversal of the serial structure. This is achieved by maintaining appropriate ordering both within the block and across block boundaries.

An alternate procedure is to look for free space in the following block prior to executing a block split. If free space is found, the last logical element in the block can be moved to the succeeding block to make room for the new logical element. If bi-directional pointers are maintained, any free space in the previous block may also be used.

Figure 4-4 illustrates the insertion of logical data elements into an ordered serial structure. The structure consists of fixed length logical elements blocked into a logically ordered record structure with a maximum capacity of three logical data elements per physical record (Figure 4-4 (a)). Block splitting is required in (b) to make room for the new element; but (c) does not require block splitting because the new element can be inserted into the free space already existing.

*It has been found that where block splitting is required to make room for insertion, storage efficiency is improved when half the contents of the block to be split is moved to a new block thus distributing the free space.



NOTES -

- (1) SHADED AREA DENOTES UNUSED FREE SPACE.
- (2) RECORD CONTROL INCLUDES:
 - (A) A LOGICAL DATA ELEMENT COUNTER
 - (B) A POINTER TO THE NEXT PHYSICAL RECORD.

Figure 4-4. Insertion for a Physically Ordered Serial Structure of Fixed Length Elements Blocked Into Logically Ordered Records With Three Logical Data Element Capacity

Logical ordering of physical records removes the need for both overflow areas and data movement.

4.6.2 File Limits

Sequential, sort oriented, access methods can be modified to recognize a set of relative numbers (file limits) identifying one or more segments of the serial structure. The set of numbers may define a series of disjoint subsets of the serial structure.

In this case, the access mechanism processes only logical data elements contained in the specified subsets of the structure. All other logical elements in the structure are automatically passed over by the access mechanism. After the last logical element of the last defined segment has been processed, the next retrieval request encounters an end-of-file condition.

Figure 4-5 illustrates the FILE-LIMITS clause as provided in the COBOL Language. When used, each pair of data-names or literals associated with the word THRU (or THROUGH) represents a logical segment of the file. In the Honeywell implementation the FILE-LIMIT clause can be used only for program documentation.

4.7 APPLICATIONS

A sequential access method is available on every major computer system in the industry. Specific applications can generally be grouped into two types of processes, external media and internal core storage media.

External media are typically tape, card, or disk files which must be passed to extract any information segments. Examples of data stored on this media are personnel files, payroll files, and history files. Files of this type are generally scanned from start to end, to either update all information or to modify one set of elements. In the latter case data is copied on an output medium until the required data element is located. At this point the modification is made. Copying is continued until either another data element must be modified or the end of the input stream has

FILE-CONTROL

Format 2

FILE-CONTROL.

```

{ SELECT [ OPTIONAL ] [ OVERLAY ] file-name-1 [ RENAMING file-name-2 ]
  ASSIGN TO file-code-1 [ For { CARDS
                               LISTING
                               MULTIPLE REEL } ]
    [ RESERVE { integer-1
                NO
              } ALTERNATE [ AREA
                              AREAS
                            ] FOR BLANK COMMON ]
    [ { FILE-LIMIT IS
        { data-name-1
          literal-1
        } { THRU
            { data-name-2
              literal-2
            }
          { FILE-LIMITS ARE
            { data-name-3
              literal-3
            } { THRU
                { data-name-4
                  literal-4
                } { THROUGH
                    ...
                }
            }
          }
    ]
    [ ACCESS MODE IS { SEQUENTIAL
                        { RANDOM
                        }
    ]
    [ PROCESSING MODE IS SEQUENTIAL ]
    [ ACTUAL KEY IS data-name-5 ] .
    { SELECT [ OPTIONAL ] [ OVERLAY ] file-name-3... }

```

NOTES

See Reference [I, File Control Paragraph] for full interpretation of the various key words, phrases, and clauses.

Each pair of data-names or literals associated with the word THRU (or THROUGH) represents a logical segment of the file.

Figure 4-5. File Limits Clause as Provided by COBOL

been reached. Often a "master-detail"* relationship is established in these structures by placing the detail records immediately behind the master record to which they relate. Positioning of these detail records is either physical or logical depending upon the ordering mode of the file.

The differences between physically ordered and logically ordered structures are very evident in processing these types. For example, a tape file is, by nature, a physically ordered file which imposes certain restrictions. These have been enumerated in Paragraph 4.5 (Disadvantages). One possible way to circumvent these restrictions is to include free space within the file itself. This approach will be discussed in much more detail in Part II. When logically ordered files are considered, the problem of insertion becomes much easier and later examples will show ways in which this can be accomplished.

There are many examples of physically ordered file processing in the H6000 architecture. Current file processing with the Time-Sharing Editor Subsystem uses this technique to insert lines of data or make modifications. Conversion of ASCII files to BCD files (or vice versa) requires similar processing of input files. Many types of history journals kept for processors such as Integrated Data Store (I-D-S), Indexed Sequential Processor (ISP), and Transaction Driven System (TDS) are serial tape files which are processed in the method previously described.

The internal storage media encompass the other types of data structures: stacks, queues, and deques. There is no requirement which dictates that these methods be restricted to internal forms of storage. However, stacks, queues, and deques are usually small data structures which need to be accessed quickly.

There are many examples of these types of data structure, especially within operating systems software. Depending upon the type of hardware, some systems will tend towards one particular technique or method. Some computer systems process stacks within the hardware itself, thereby making them much more powerful and convenient. Burroughs large-scale operating systems are driven by hardware stacks

*In this context the records are sometimes referred to as a header record followed by its associated trailer records.

which enhance the power of the operating system and language processors such as ALGOL. The DEC PDP-11 uses a stack to process interrupts with minimum software/hardware overhead. The Honeywell H6000 does not have the hardware to expedite processing of stacks, therefore they are less frequently used for this system. In the H6000 system the major usage is to manipulate the instruction counter and register storage. This is normally referred to as the IC and I stacks.

Within the Honeywell structure, many of the operating systems tables are manipulated in the form of queues or restricted dequeues. These are used to facilitate processing events which must be kept in chronological order to allow system functions to proceed in an orderly fashion without adversely affecting the scheduling and execution of unrelated activities. System scheduling (.CRRGQ), peripheral allocation (.CRJOB), core allocation (.CRPOQ) and system output processing (.CRCYQ) are specific examples of queuing for interphase communication. Almost every major type of application where ordering is important will utilize some type of deque structure.

Logically ordered structures, such as I-D-S (see Figure 1-1), use the current and next record, irrespective of the physical placement upon the media. There exists a very fine distinction between sequentially processed linked lists (e.g., stacks, queues, and dequeues) and random linked lists (e.g., multi and inverted lists). Therefore, further discussion of linked lists has been deferred to Chapter 6.

Whenever a programmer wishes to organize a fixed area into small variable sized segments, linked lists (elements connected by pointers)* are usually preferred. Within most operating systems, core management is segmented into pages or blocks which represent either free or used core. When a request for more core is processed, the core manager need only to follow his pointers, serially examining all core until the desired space is found. The paging algorithm for virtual storage management depends upon this data structure to allow efficient manipulation of the real core storage within a fixed physical space.

*For other application of linked lists see Part II.

The Honeywell system utilizes logical processing to maintain flexibility within the I/O operation. All I/O queues are linked together to provide an easy and efficient mechanism to scan data elements throughout core. Since the H6000 hardware has an instruction REPEAT LINK (RPL) to expedite processing linked lists, structures of this type are easily maintained. Buffer managers such as Transaction Driven System (TDS) and Time-Sharing are common users of linked data structures.

As described in Paragraph 4.2.2, sequential methods can be utilized to gain access, based upon a defined key field imbedded in the data, to records within a given logical structure. The Honeywell I-D-S system links all records together to form chains* for efficient retrieval of records. The various chain linkage is specified by use of the CHAIN-ORDER clause of the I-D-S language. Language options allow a user to determine the methodology and techniques used to process each chain.

The pertinent options are SORTED and SORTED WITHIN TYPE. With the SORTED option, the records of a chain are ordered sequentially based upon a designated field. I-D-S permits a number of record types within one chain. With the SORTED WITHIN TYPE option, the records of a chain are sequenced within each record type. However, the records are not grouped by record type. Figures 4-6 and 4-7 illustrate the SORTED and SORTED WITHIN TYPE options.

*Chain and linked-list are generally used interchangeably. Any difference is more one of emphasis.

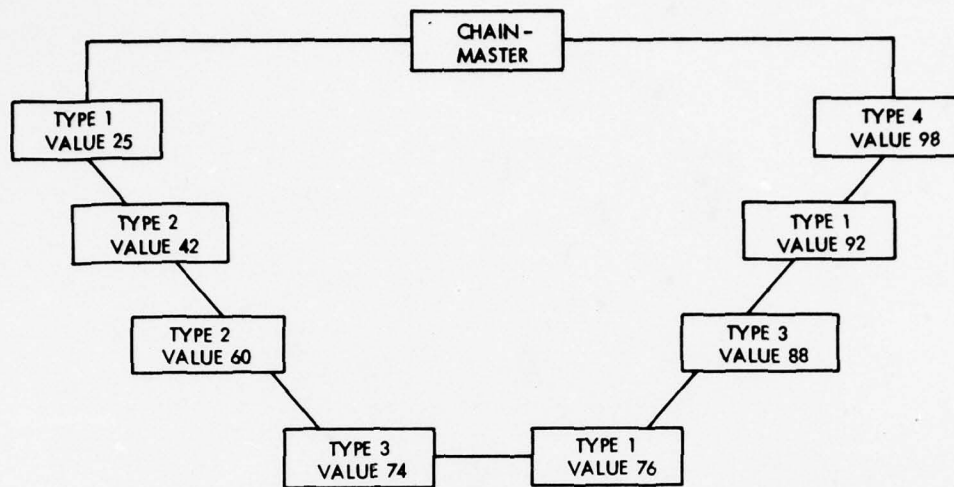


Figure 4-6. Chain-Order Sorted

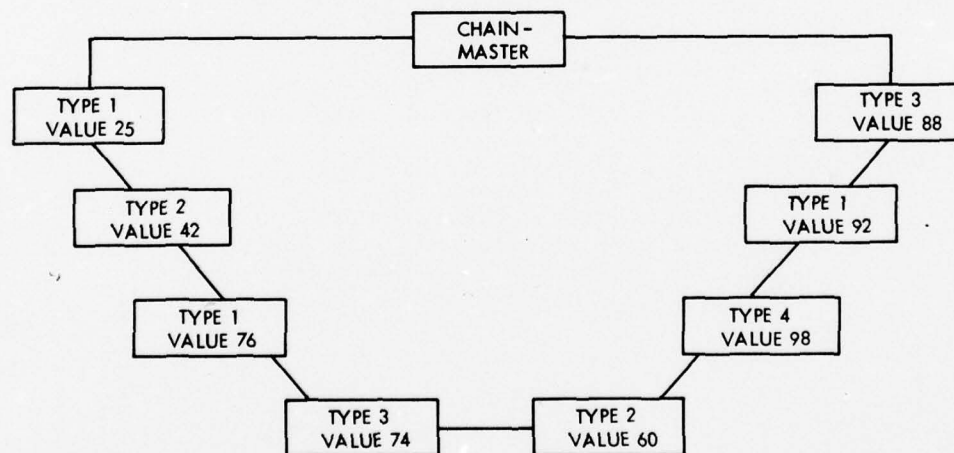


Figure 4-7. Chain-Order Sorted Within Type

CHAPTER 5 - RANDOM ACCESS OF SERIAL STRUCTURE

5.1 GENERAL

Random access methods usually insert, modify, delete or retrieve specified logical data elements directly. If the specified element is nonexistent, an error condition is shown. A user does not have to sequentially traverse the logical sequence until a desired element is located, but can go to it directly.

Random access techniques are usually applied to serial structures for which the linear relationships between logical elements can be determined apart from accessing mechanisms and associated algorithms. Hence, Figures 3-2 and 3-4 illustrate the types of serial structure to be discussed in this chapter.

5.1.1 Insertions/Deletions

If logical relationships are maintained through physical positioning, as illustrated in Figure 3-2, every insertion and deletion will require physical movement of the data within the data structure to express the new logical relationships. If logical relationships are maintained through pointer linkage, as illustrated in Figure 3-4, only adjustments in the effected pointers are required to express the new relationships, and the insertions may be stored in the next available location in the data space.

5.1.2 Flagging

Since this chapter deals with serial structures and associated access methods, the flagging described in Paragraph 4.1.2 applies.

5.1.3 End-of-Space Condition

Again, since this chapter discusses access of serial structures, the physical data space descriptors and end-of-space condition described in Paragraph 4.1.3 apply here as well.

5.1.4 Element Formats

Random access methods generally accommodate both fixed and variable length formats as described in Paragraph 4.1.4.

5.2 ACCESS METHODOLOGY

Random access methodology applicable to serial structures centers around either a relative sequence number or a defined key.

5.2.1 Relative Sequence Method*

To provide random selection of logical data elements on the basis of an element's logical position in a linear list which constitutes a serial structure requires that, at any point in time, a unique identifier is associated with each element of the list and that this identifier is known to both the user and the random access mechanism. The unique identifier is known as a relative sequence number. It is defined as a number which denotes a logical element's position in the list and is expressed as a logical displacement from the first element of that list.

The random access mechanism operating on a serial structure using relative sequence referencing employs:

1. Sequence control containing the relative sequence number for the current logical element
2. Current logical element pointer
3. Data space delimiters.

There are many different ways of implementing a relative sequence method. A straightforward approach keeps the sequence control current with the current logical element pointer and takes into account all modifications of the structure since its last reorganization. The current logical element pointer is used to optimize movement to a new specified logical element or provide a sequential capability from a last specified position.

*Actual physical addresses can be used but then the mechanism becomes device dependent which limits operational flexibility. The I-D-S DIRECT method utilizes hierarchical logical address pointers.

It provides for:

1. Initialization of accessing controls
2. Random access to logical elements of the list using specified relative sequence numbers.

It is the user's responsibility to maintain a record of sequence numbers for the logical elements he may wish to address randomly. Consequently, he must be cognizant of all the updates of the data base, because every insertion and deletion alters the relationship between the sequence numbers and the logical elements of the structure from the point of insertion or deletion to the end of the structure. If flagging is employed, only cognizance of insertions is required, since logical deletions do not otherwise affect the data structure.

Should a user's record of sequence numbers become partially damaged, it can be readily restored via a sequential pass through appropriate segments of the data structure. Random access capability can be used to move to the beginning of each segment, after which sequential access is used within each segment.

5.2.1.1 Physical Ordering

If the serial structure is physically ordered (Paragraph 3.2) with fixed length logical elements, the location of any element may be computed directly through evaluation of a relatively simple algebraic expression such as:

$$\text{LOCATION} = (\text{RSN}-1) \times \text{LENGTH} + \text{FIRST}$$

where

RSN = relative sequence number identifying the logical element to be located

LENGTH = length of logical element

FIRST = location of first element of linear list

LOCATION = location of desired logical element.

When more than one data space description is associated with the serial structure, additional parameters will be required to define the physical structure and/or device.

If the structure does not contain fixed length logical elements or a suitable expression cannot be evolved, sequentially "stepping" through the linear list may be necessary.

When a specified sequence number is greater than the current value of the sequence control, stepping starts with the "current" logical element and continues in the forward direction until the desired element is located or an end-of-data condition is raised. When the specified sequence number is less than the current value of the sequence control, stepping starts with the first element of the list and continues forward until the specified element is located or an end-of-data condition occurs, or it may proceed backward from the current position until the specific element is located or a beginning-of-data condition is encountered.

5.2.1.2 Logical Ordering

If the serial structure is logically ordered through use of appended pointers (Paragraph 3.3), the specified logical element is usually located via "stepping" as described for physical ordering of variable length logical elements (Paragraph 4.2.2). If the appended pointers are bi-directional, backward stepping may begin with the current logical element.

5.2.2 Defined Key Method

To provide random selection of logical data elements on the basis of the value of a defined key field imbedded in the logical data elements in a linear list which constitutes a serial structure, it is necessary that the defined key field is known to the random access mechanism, and the set of values which identifies the logical elements to be randomly addressed are known to the user.

The random access mechanism operating on a serial structure using a defined key field imbedded in the logical elements utilizes:

1. Current logical element pointer
2. Data space delimiters.

The current logical element pointer is used to facilitate movement to a specified next logical element or to provide sequential continuation from the last specified position. It provides:

1. Initialization of accessing control
2. Random access to logical elements of the list using specified key values.

It is the user's responsibility to maintain record key values for logical elements he may wish to address randomly. Should a user's record of key values suffer damage, it can be readily restored via a sequential pass through the appropriate segments of the data structure. Random access capability can be used to position to each segment. Sequential access would be used to and through the respective segments.

5.2.2.1 Physical Ordering

When a serial structure is physically ordered by fixed-length logical data elements (Paragraph 3.2), the location of any element can be found through use of a binary search.

Binary search is a searching technique for a linear list of elements sorted according to the collating sequence in a defined key field. To do this, one proceeds to the center of the area to be searched and compares the key of the record located there with the specified search key. By this process, the search area is reduced by half. The process is repeated until the desired element is found.* The data space will be inspected approximately $\log_2 N - 1$ times on the average, where N is the number of elements in the linear list to be searched. Figure 5-1 illustrates the technique.

*There are many methods used to bias the selection of the "mid-point" in order to take full advantage of known distribution/characteristics of the data to be searched.

	STORAGE ADDRESS	DATA	RSN
	105	EMP # 115	1
	110	EMP # 120	2
	115	EMP # 130	3
	120	EMP # 175	4
SECOND INSPECTION 275 > 210	125	EMP # 210	5
	130	EMP # 250	6
FOURTH INSPECTION 275 = 275	135	EMP # 275	7
THIRD EXPOSITION 275 < 290	140	EMP # 290	8
	145	EMP # 300	9
CURRENT ELEMENT POINTER AND FIRST INSPECTION 275 < 325	150	EMP # 325	10
	155	EMP # 350	11
	160	EMP # 375	12
		...	13

NOTE

SPECIFIED KEY VALUE IS 275.
 CURRENT RELATIVE SEQUENCE NUMBER IS 10.
 CURRENT LOGICAL DATA ELEMENT IS EMP # 325.
 RSN DENOTES RELATIVE SEQUENCE NUMBER.
 CURRENT START ADDRESS IS 150.
 LENGTH IS 5.

Figure 5-1. Illustrative Example of Binary Searching a Physically Ordered Linear List of Data Elements

The location of the next mid-point can be computed directly through evaluation of a relatively simple algebraic expression such as the following:*

If $K < K_i$

$$\text{LOCATION} = (1/2(\text{CRSN} - \text{LRSN}) \times \text{LENGTH}) + \text{START}$$

If $K > K_i$

$$\text{LOCATION} = ((1/2((\text{CRSN} - 1) - \text{HRSN}) + \text{HRSN}) \times \text{LENGTH}) + \text{START}$$

where a sequence control includes the CRSN, LRSN, and HRSN, and

K = specified key values
 K_i = key values for current logical data element
 CRSN = RSN for current logical element
 LRSN = RSN for current low logical element
 START = address of beginning of list
 HRSN = RSN for current high logical element
 LENGTH = logical element length
 LOCATION = address of next logical element to be inspected.

A binary search is generally not practical for searching direct access storage devices because of the time required, nor can it be used for searching a linked list, since a next mid-point cannot be computed. It is primarily useful for searching main memory or solid-state storage devices.

When either the serial structure contains variable length logical elements, or a binary search is not suitable, the simplest way to locate a logical element is to scan the linear list, inspecting the key of each element. One can start at either the "current" logical element, or the first element of the list, depending on the value of the specified search key in relation to the value of the key for the current element. When there is only an occasional need for random access, a serial scan may suffice. However, when a significant amount of processing requires random access followed by a serial scan, a random structure with sequential access capability as described in Part II would be more suitable.

* Integer arithmetic is truncated.

5.2.2.2 Logical Ordering

When the serial structure is logically ordered (Paragraph 3.3) with fixed or variable length data elements, the simplest way for locating a logical element is to scan the linear list key by key for each element. One can start either at the "current" logical element or the first element in the list based on the value of the specific search key in relation to the value of the key of the "current" element.

Pre-ordering of search keys in the same sequence as the logical elements of the serial structure minimizes scanning time. As noted in Paragraph 5.2.2.1, a user should evaluate the use to which a serial structure is to be applied. If a significant proportion of the references to the structure involves random access, the structure should optimize the random capability.

5.2.3 Indexed Sequential Method

Indexed sequential access of files depends upon a key and an index. A key is simply a data field of a record that is used to identify that record, and the key must be the same field in each record of the file. An index to a file is a table containing the keys of the records in that file. When a record is to be accessed, the given key is found in the index and the physical location of the record is determined.

5.2.3.1 Key

Each record of a file must have a key that distinguishes that record from every other record of the file. This unique key is known as the primary key and is used to identify the record. The key may be a single data field of the record or it may be a concatenated key, composed of two or more consecutive data fields. A record may also have secondary keys which are not necessarily unique. Such a key is a generic key which is usually used to describe some attribute of the record. Record retrieval may be through use of a secondary key, but the record is always stored based on its primary key.

A file with records which have only a primary key is known as a single-key file. If the records have at least one secondary key, the file is called a multi-key file.

5.2.3.2 Index

An index is a table containing keys and pointers. A full, or dense, index is one which contains every value of a particular key and a pointer to each record. A partial, or sparse, index contains only a subset of all values of a given key. The pointers in a partial index generally point to a block, page, or bucket which may contain any number of records.

A pointer may be one of several types. It may contain the location of a record or it may point to lower level indices. A pointer may indicate a bucket location or it may point to the head of a chain which contains the desired record.

An address specified by an index may be an absolute machine address, a relative address, or a symbolic address. The absolute address is the actual location of a record on its storage device. This address must eventually be obtained but it is usually determined by a lower level index, rather than the one initially accessed. A relative record address specifies the location of a record in relation to the beginning of a particular block or page of records. Rather than specify an absolute or relative address, an index may give a symbolic pointer for a record. In this case, another technique (e.g., hashing) must be used to determine the record's actual location.

An index may be either primary or secondary. A primary index contains the primary keys of a file and a secondary index contains secondary keys. Most data base management systems utilize a combination of a primary index and several secondary indices. A primary or secondary index may be either partial or full.

5.2.3.3 Index Access Method

Many techniques are available for accessing random files using the indexed sequential method of access. An index is always used; however, it may be a serial or a random structure. Data is initially accessed by using the index, but additional records may then be accessed sequentially, starting with the current record.

Two important considerations for the indexed data access are the organization of the index and the organization of the data. The index contains the keys which are used to locate the data. Each record must have a key, but the index does not necessarily contain all of the keys of a given file. In most cases, a number of records are stored sequentially in a block, and the index will contain the highest key in each block of the file, arranged in ascending order. When the index is accessed and the appropriate block located, that block is brought into main memory. The block is then scanned to find the record containing the desired key.

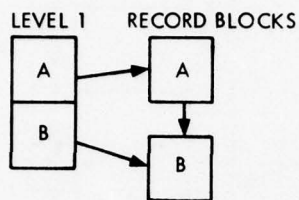
Rather than specify the location of a record, an index may point to a lower level index. Any number of levels may be used, either preset or dynamic. Figure 5-2 (b) illustrates two levels of index. Figure 5-2 (c) illustrates n levels with bottom-up fine-to-coarse (level 1 to level n) dynamic construction before the data is accessed. The methods of accessing an index will apply to each level. However, the same method is not required at each level. It should be noted that the data blocks are linked together to support standard sequential accessing. Logical ordering of the record blocks (Paragraph 4.6.1) within each level can facilitate maintenance, and a balanced bottom-up construction (Figure 5-2, a-c) can provide optimal overall performance.

If the index is serially structured, it may simply be an unordered list or it may be ordered sequentially by key values.

5.2.3.3.1 Sequential Access

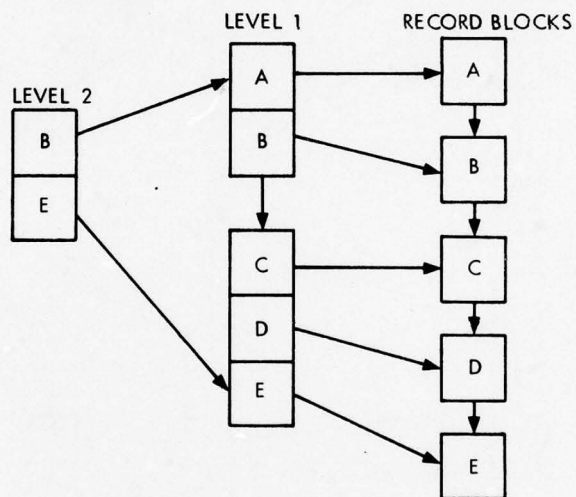
In an unordered list the only means of locating the desired key is by scanning the index until it is found. If the key is not found, the desired record does not exist. When searching a large index this method may become inefficient, but can be useful for searching a portion of a large index.

When the index is ordered sequentially, the desired key may be found by using the binary search method, as described in Paragraph 5.2.2.1. To be efficient, a binary search requires that the entire index is in main memory. If the index is ex-



DATA BASE CONTAINS ONLY TWO BLOCKS.

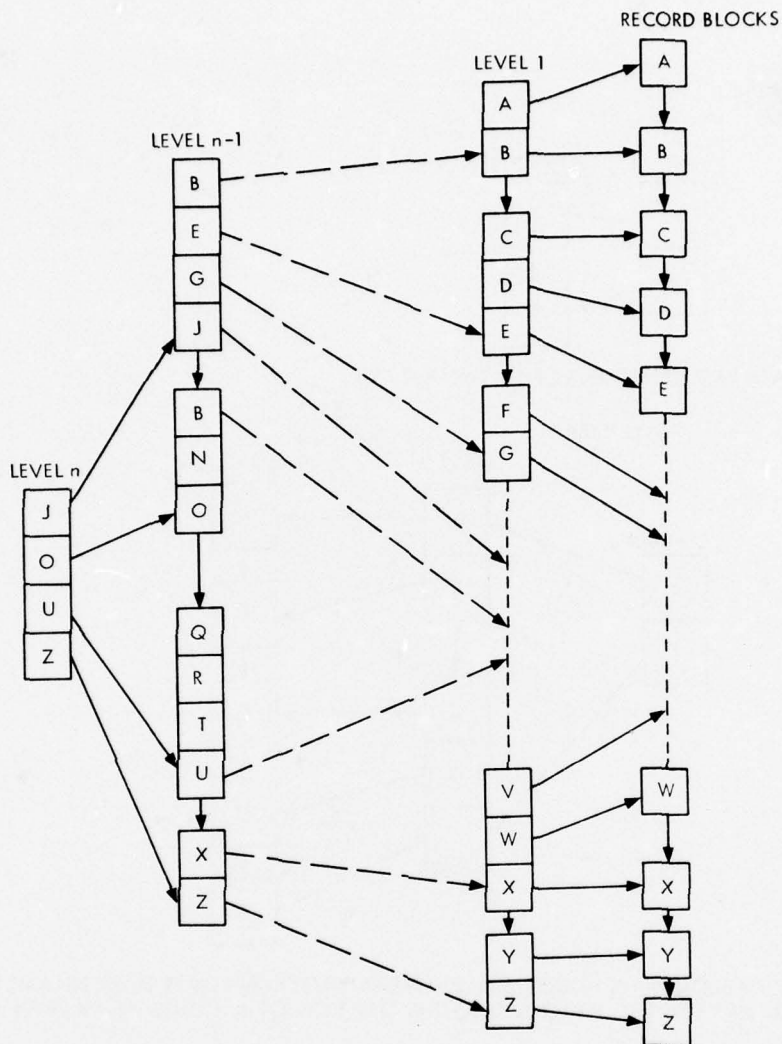
(A). STEP 1



THREE MORE RECORD BLOCKS ARE ADDED (SINCE THE FIRST BLOCK IN LOWER LEVEL BECAME FULL, A SECOND BLOCK WAS STARTED, WHICH LED TO THE CREATION OF A HIGHER INDEX LEVEL.)

(B). STEP 2

Figure 5-2. Levels of Index (1 of 2)



INDEX BLOCKS MAY BE LINKED (COMPACTED) TO FACILITATE MAINTENANCE PROCESSING.
DATA BLOCKS ARE LINKED TO SUPPORT FIFO SEQUENTIAL ACCESS OF THE FILE.

(C) STEP 3.

Figure 5-2. Levels of Index (2 of 2)

tremely large, or resides on some storage device, a binary search can be time-consuming.

A block search may also be performed on a sequentially ordered index. In this case, the index is subdivided into blocks and the highest key in each block is checked. The search starts at the beginning, or lowest value, of the index. When a key is found which is higher than the search key, that entire block is scanned until the desired key is found. If the high value key in a block is equal to the search key, no scanning is necessary.

Each access of the index is called a probe. The optimum size for a block search is the size which requires the minimum number of probes and is $\lceil \sqrt{N_k} \rceil$, where N_k is the number of keys in the index. ($\lceil n \rceil$ denotes the smallest integer larger than n , if n is not an integer.) A block search (Figure 5-3) would be appropriate for a small index or for searching portions of a large index.

5.2.3.3.2 Random Access

Scanning, binary search, and block search are sequential methods of accessing a serial index. However, a serial index may also be accessed in a random manner. If the relative sequence number of the desired key is known, the corresponding position in the index may be accessed directly. This technique is described in Paragraph 5.2.1 in connection with accessing records, but it also applies to accessing items within an index.

5.2.3.4 Data Blocking

In an indexed sequential file, records are stored sequentially in blocks, and an index contains the highest key of each block. A block is usually a track or a cylinder on a direct access storage device. A block, or track index contains the highest key for each track. When a file spans more than one cylinder, a cylinder index is kept which contains the highest key of each cylinder. If a block is a group of words or characters whose size is equal to an I/O segment of peripheral devices, then paging techniques can be used to achieve a degree of device independence.

AD-A041 459

COMPUTER SCIENCES CORP FALLS CHURCH VA

F/G 9/2

A COMPREHENSIVE DATA BASE ACCESS METHODOLOGIES DESIGN GUIDE.(U)

SEP 76 J EMORY, S GREEN, R GRIMES, O HASLOP

DCA100-75-C-0029

UNCLASSIFIED

CSC-R493700019-2-2

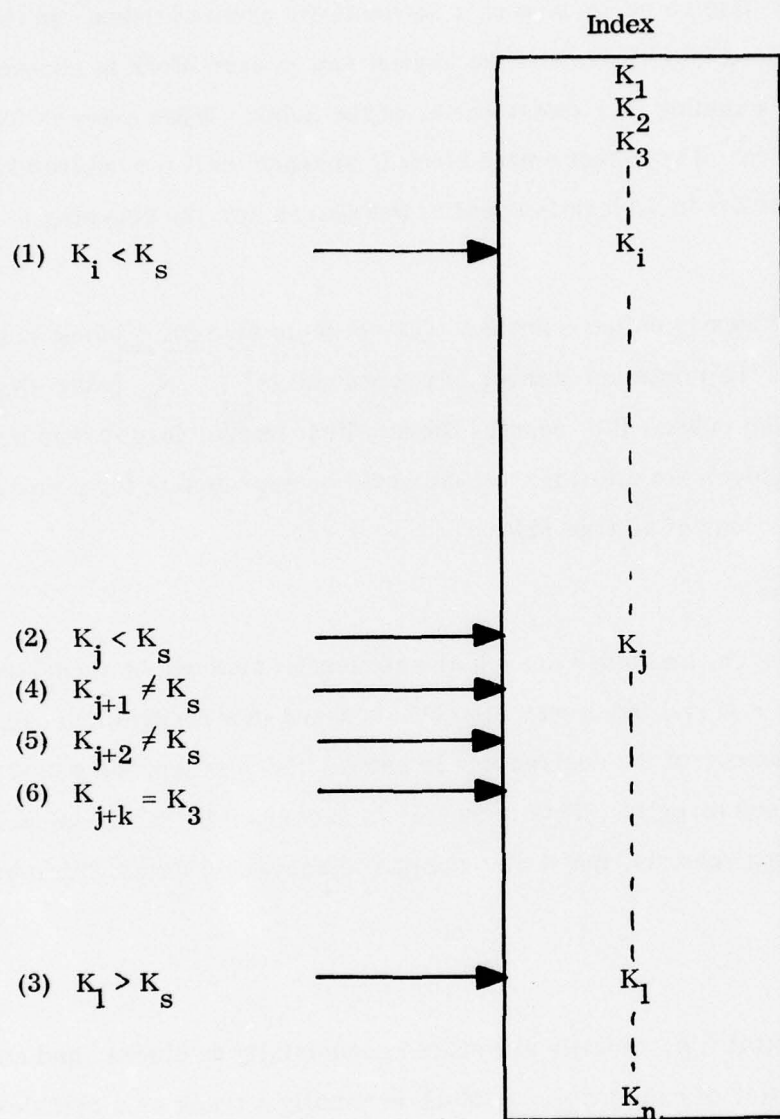
CCTC-TM-123-76

NL

2 OF 4

AD
A041459





Index is Ordered Sequentially with $K_1 \quad K_2 \quad K_3 \dots \dots K_n$

Figure 5-3. Block Search for Key K_s

5.2.3.4.1 Insertion

To add a new record to the file, its key is compared to the keys in the index and the appropriate block for storing the record is located. The new record is then inserted into its proper position in the block. Another record may have to be moved to make room for the insertion and this procedure may cause records to be moved into an overflow area. It should be noted that if logical blocking of records is utilized, and overflow records are "split" as described in Paragraph 4.6.1, then an overflow area is only required when all of the available blocks in the base file have been allocated. Additionally, the overflow area is merely a simple extension of the base file, and overflow handling as discussed in Paragraph 5.2.3.4.2, is generally simplified.

5.2.3.4.2 Overflow Records

An overflow area may reside on the same cylinder as the primary storage area or it may reside as an independent area on another cylinder. To access records in an overflow area, several methods can be used.

A pointer in the track index can refer to the head of a chain of overflow records. The pointers throughout the chain indicate the correct key sequence of the records. The overflow records may be stored in the primary storage area or in an independent area. If an independent area is used, much seek time may be required to follow the chain.

Another method of accessing overflow records is to organize the track index so that it can contain multiple pointers. Since one pointer is required for each overflow record, the track index may become exceedingly large.

Overflow records can be stored in an independent area that has its own index. Any index of an overflow record will require location of the overflow cylinder and a search of the overflow index. However, the traversal of chains is eliminated.

To decrease the number of overflow records, and thus reduce search time, distributed free space may be allocated. When a file is initially created, the blocks or tracks are only partially filled (usually 75 percent). The unused space is used to

accommodate insertions. A new record is simply placed into the unused space, or the old records are moved so that the new record will appear in proper sequence. Sequencing is preferable because search time for the record is reduced.

A file organization utilizing distributed free space should also be capable of handling overflows. After many insertions, the free space may become filled and overflow space will be required. In fact, an overflow area can itself overflow and be extended to another area. In such a case, the file should be reorganized. The index must also be updated to reflect the reorganization of the file.

5.2.3.4.3 Deletion

The deletion of records from an indexed sequential file is a trivial matter. A record is simply marked, such as the setting of a delete bit, to indicate the record's deletion. When the file is reorganized, the record is then physically deleted from the file. However, in the meantime, before the physical reorganization occurs, a new record can be added to the file by writing over the record to be deleted. This overwrite can occur only if the new record is the same size or smaller than the deleted record and if the new record is in the correct key sequence.

An indexed sequential file organization can handle either fixed or variable length records. However, the records are stored in fixed length blocks. Each block is preceded by a control word (or words) indicating the number of words which have been used or the next position available for storage. The control word may also contain a pointer to the overflow area.

5.2.3.5 Sequential Access Method

Records in an indexed sequential file may be accessed sequentially, as well as randomly, on the primary key. In sequential processing, records are accessed in key sequence, since that is the order in which they were stored. Records are often accessed initially in a random manner and then searched sequentially from that point in the file.

5.2.3.5.1 Variable Length Records

Each record in an indexed sequential file is preceded by a record control word which specifies the number of words in the record. This information is essential for determining the start of the next record. The record control word is used primarily for variable length records and need not be included in a file of fixed length records.

5.2.3.5.2 Pointers

Each record in the file will also contain a pointer to the next sequential record. The pointer will indicate a block number and the location within the block of the next sequential record. The next record may be the next one in the block, the first one in the next block, or one in an overflow area. Wherever this record resides, it will also have a pointer to its succeeding record. The pointer in the last record of the file will indicate an end-of-file condition.

With the information contained in the record control words and in the pointers, an indexed sequential file can be accessed efficiently.

5.2.4 Inverted List File

Another type of file organization that depends upon keys for record access is the inverted list file. In this file, records are usually accessed on the basis of one or more attribute values. Thus, a record may have any number of secondary generic keys, in addition to its unique primary key.

5.2.4.1 Inverted Lists

With inverted list files, an additional file, or table, is required between the index and the data file. The table is composed of lists, known as inverted lists. Each inverted list contains pointers to the data records. The index, in the inverted list file, is composed of generic keys. Each key points to a list of all records containing that key. The pointers in a list may be either the addresses or the unique keys of the desired records. If an address is given, it may be a block address, designating the block to be scanned for records containing the given key. Alternatively, an address may indicate the block address plus the record within the block of the desired record. Figure 5-4 illustrates the inverted list file organization.

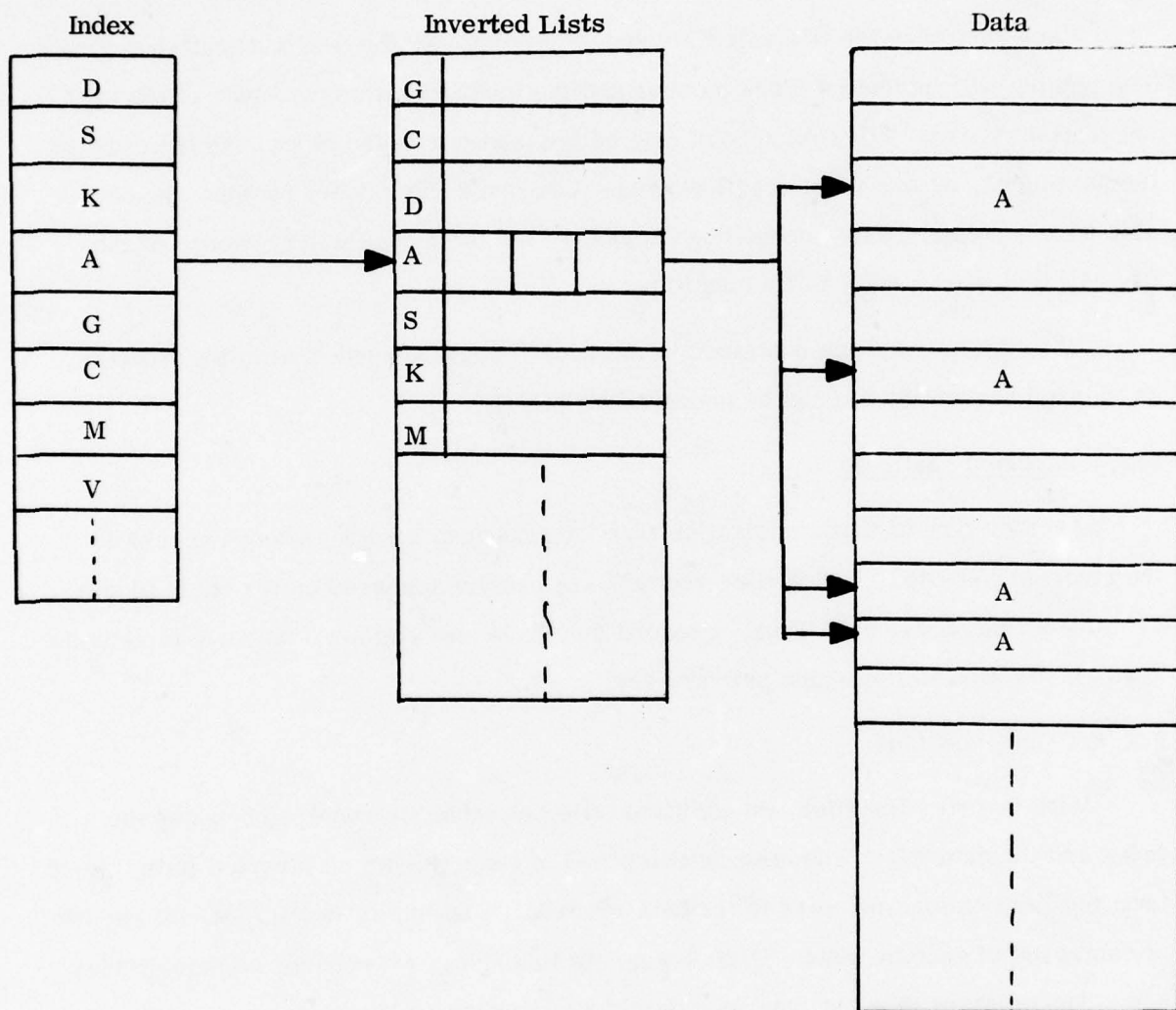


Figure 5-4. Inverted List File Organization

5.2.5 Access by Secondary Key

Record access based upon a secondary key is a desirable feature of a data base management system. One method which provides such access is the inverted list file organization. An index and corresponding inverted list can be kept for each secondary key of the file, thus facilitating record access by each key.

A file with many secondary keys may require a large amount of storage space for inverted lists. An alternate method is the use of bit maps, or bit vectors, as will be described in Paragraph 6.3.4. For each key, one bit map can be generated for each record in the file. Each bit corresponds to one key value, indicating whether or not that record contains the given value.

Bit vectors may also be used on a block basis; e.g., a one-bit indicates that the block contains a record with the given key values. The block must be scanned to find the record(s) containing the key.

5.3 ACCESS CONSIDERATIONS

In general, techniques for binary searching of physically ordered serial structures are efficient for main memories and solid-state storage but are not suitable for searching direct access storage devices, because of time consuming multiple accesses. They are also not used to search linked lists because at present no method exists for calculating the address of the next mid-point to be tested. Sequential inspection of the key of each logical element of a serial structure is usually too slow to be acceptable except for applications which use the random capability infrequently and applications where the interval to the specified element is relatively small. The method can be readily adapted to process partitioned data files such as program libraries and serial structures stored on magnetic tape.

An important aspect in the design of an indexed sequential file is the selection of the primary key. A record in the file may be accessed by more than one key, but the primary key will be used to determine the appropriate block for storing the record. Thus, the records will be sequenced based upon the primary key.

The key chosen as the primary key should be the one most frequently used in referring to the records. The primary index although containing primary key values, will usually be only a partial index. Any number of secondary indices may also be generated. However, each secondary index must be a complete index.

For most efficient access, the primary index should reside in main memory. If the index should become extremely large, the primary index should be reorganized and a first level index created. The additional index can be stored on a direct access storage device. Each record access will now require two seeks, one for the first level index and one for the block containing the desired record. However, the amount of main memory required will be greatly reduced.

5.4 ADVANTAGES

The principal advantages of randomly accessing serial structures, in addition to those enumerated in Paragraph 4.4, are: the data can be serialized and accessed using the method most appropriate for a majority of the applications; and a limited capability is provided for others who require access to the structure.

The primary advantage of the indexed sequential file is the capability to access the files in either a random or sequential manner. For example, a number of users may use a particular file. Each may access the records in the manner most suitable for his application and also alternate between random and sequential access at any time.

Another advantage of the indexed sequential organization is realized when additions are made to the file. The index to the file is almost always a partial index. Hence, when new records are inserted, the index seldom requires modification.

5.5 DISADVANTAGES

The principal disadvantages associated with randomly accessing serial structures are:

- Updating a physically ordered data structure requires physical movement of data within the structure which often requires a rewrite of the entire structure

- Binary search techniques have primary applicability for physically ordered serial structures stored in main memory, or on a solid-state device, in which the search area is one contiguous block
- Techniques which sequentially inspect each key of logical data elements are too slow for most purposes and are usually limited to batch processing applications
- Pre-ordering of records to be updated is required to optimize sequential processing.

Adding records to an indexed sequential file may become cumbersome if a large number of additions are required. When a new record is inserted into the appropriate block, records currently in the block may have to be moved to make room for the insertion and an overflow may occur. When the overflow area becomes filled, it is time to reorganize the data file and update the index.

5.6 VARIATIONS

Blocking and file limit techniques described in Paragraph 4.6 are equally applicable to random and sequential access of serial structures. Random techniques using directories and indices can be applied to locating physical records and/or logical data elements in a serial structure. For specific details applicable to indexed-sequential access methods, see Chapter 7.

5.7 APPLICATIONS

Methods for randomly addressing serial structures are available for every major computing system in the industry. In particular, binary searching is widely applied to indices and directories. This technique is used more often to search such indices and directory files than searching the data files to which they refer.

Processors which provide the capability for indexed sequential file organization have been developed by most computer manufacturers. These processors are extensively utilized for a variety of applications.

Honeywell's Indexed Sequential Processor (ISP) provides for the creation, access, and maintenance of indexed sequential files. A utility program is provided to unload and reload files. Records to be deleted from an indexed sequential file are simply marked for deletion. When the file is unloaded from mass storage, the deleted records are ignored. When the file is reloaded, a new index is created and the data records redistributed throughout the file.

ISP also provides for journalization and restoration. When a file is being updated, each page that is changed is written on a journal tape as it appears before and after the change. (In Honeywell terminology, a page is synonymous with a block.) Journal tapes are used to restore a file to its original condition either before or after an update.

Distributed free space in a file is obtained by specifying a percent-fill when the file is created. ISP places a number of records on each page (block) so as not to exceed the percentage specified. The remaining space is reserved for later insertions. In addition, separate overflow space is allocated for each file.

ISP permits variable-length records, but the page or block size must be specified when an indexed sequential file is created.

IBM has developed an Indexed Sequential Access Method (ISAM) for processing indexed sequential files. In creating a file, ISAM stores the records in blocks so that one disk sector or track is equivalent to one block. Any records later added to the file are stored in an overflow area.

ISAM uses two overflow areas. Initially, overflow records are stored in an overflow area on the same cylinder as the primary data area. When this area becomes filled, additional overflow records are stored in an independent overflow area on a separate storage unit.

IBM's Virtual Storage Access Method (VSAM) also provides an indexed sequential technique which is hardware independent. Storage for VSAM files is based on control areas and control intervals, rather than cylinders and tracks. A

number of records may be stored in one control interval, but the size of the interval is fixed for each file and must be specified when the file is created.

VSAM uses distributed free space to accommodate record additions. When a file is created, some free space is left in each control interval. Additionally, some control intervals within a control area are left empty. For each empty control interval, a free space is left in the corresponding index. The amount of free space in a control interval and the number of empty intervals desired must be specified for file creation.

As will be further developed in Part II, sequential files with indices (indexed-sequential files) constitute the most common form of file addressing.

PART II

The advent of third-generation computer hardware and the need for a variety of data representations, make the serial structures sometimes appear to be inadequate. Since disks have become the primary storage medium, random structures and the techniques necessary to process the structures have assumed increased importance. Part II presents the most common structures in use today and a representative set of techniques.

Chapter 6 has been influenced heavily by the work of Martin [G].

CHAPTER 6 - RANDOM STRUCTURES

6.1 GENERAL

A random structure is defined as a structure with the property that permits any part of the data to be obtained without dependence upon the previously obtained data. The addressing technique of a random structure is based upon a specific set of rules which relate to placement and method of access. Information is represented in two ways: logical and physical. These were described for serial structures in Part I. They are equally appropriate for dealing with random structures.

To restate the difference between logical and physical organization: logical refers to the way in which a programmer views information, and physical refers to the way the system analyst sees it. The two aspects are combined in the catalog or directory which is the chief responsibility of the data administrator. A programmer is primarily interested in a subset of the total collection, a system analyst is concerned with the way information is physically stored in the system, and the data administrator is concerned with both aspects of organized information. Figure 6-1 illustrates the difference between logical and physical structures. The two are interdependent for one physical record may contain several logical records.

6.2 LOGICAL STRUCTURES

The most important question to be answered by the efficient utilization of any data base structure is "how well does the logical structure model the enterprise". If the relationships expressed in the model are in error or incomplete, the data base will not be capable of providing the desired information. The best insurance against such an occurrence is to establish an overall diagram of the data base including all entity interrelationships. Appropriate use of schemas and other descriptive language ensures that all information retrieval needs can be met.

6.2.1 Representation Standards

Schemas used to describe logical structures must provide observers with a

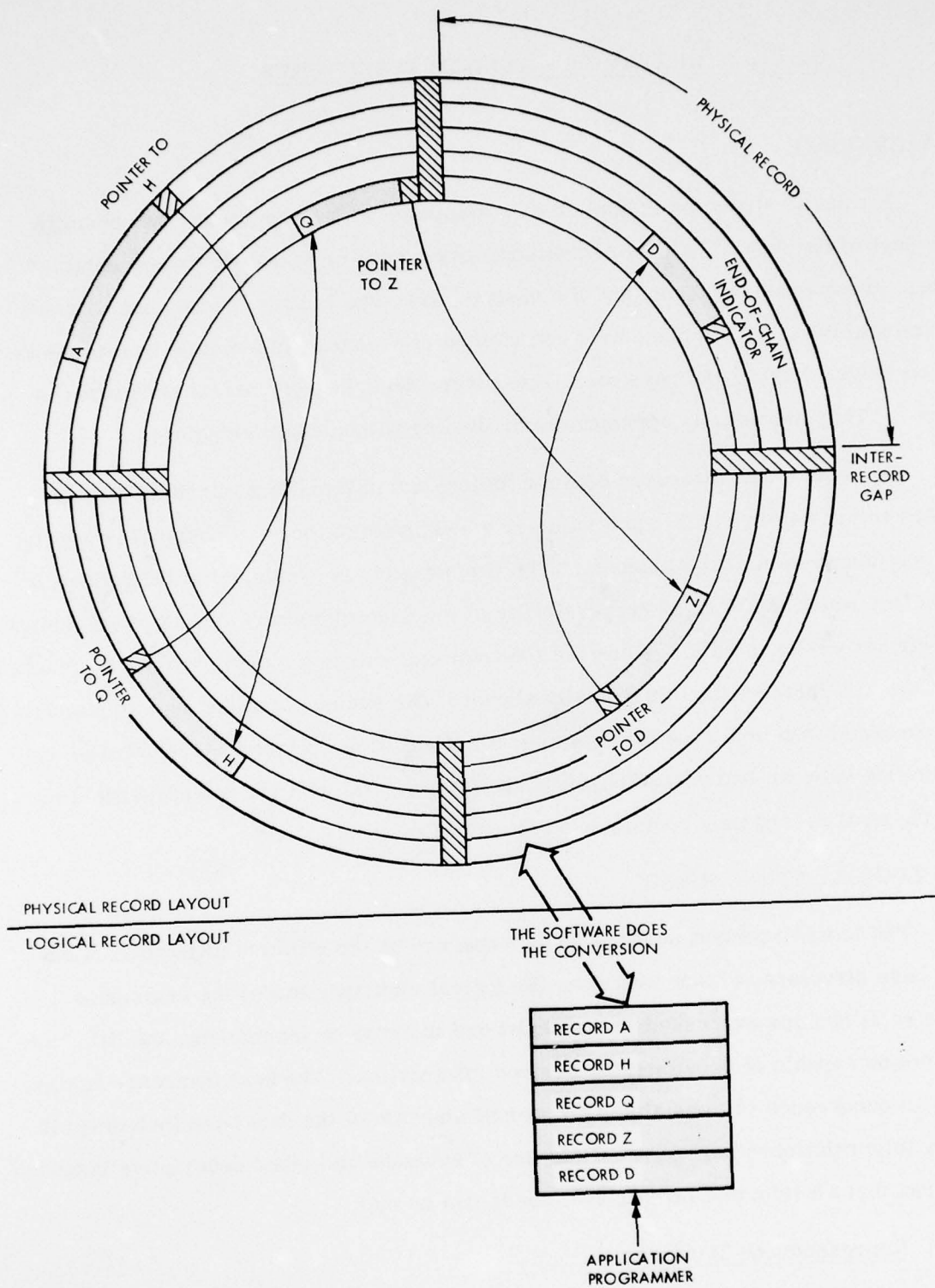


Figure 6-1. Example of Difference Between Physical and Local Data Organization

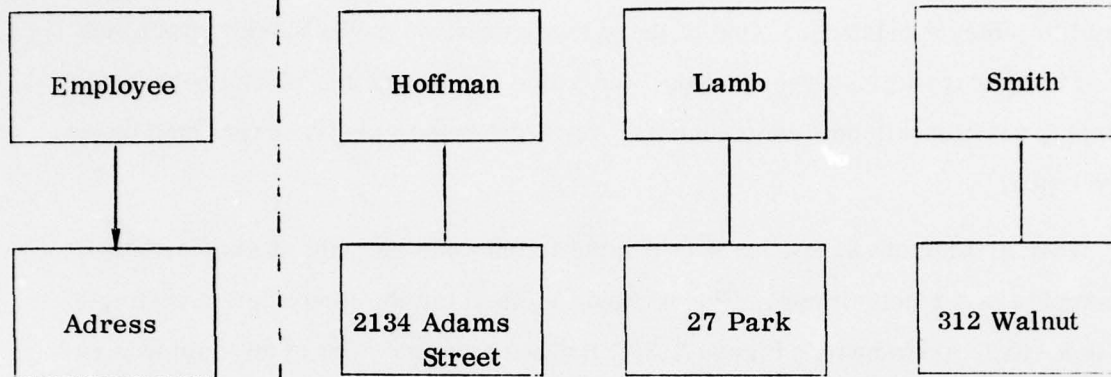
clear understanding of the structure and element relationships. This can most readily be achieved through development of standard representations.

6.2.1.1 Common Practice

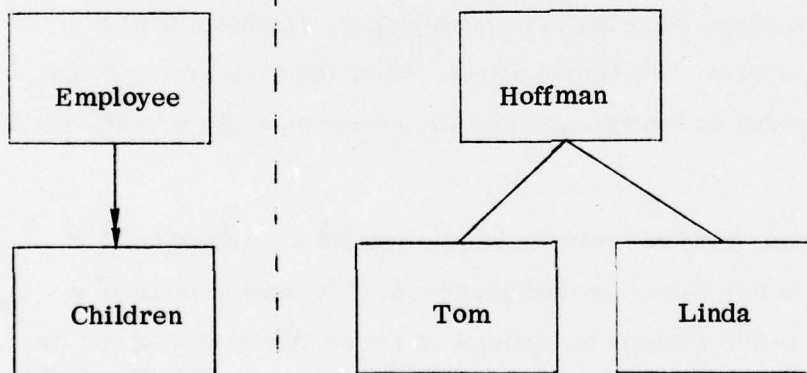
To establish a data structure shorthand, specific basic configurations are utilized to indicate relations. One of these configurations is the block. The block is used to indicate a given record type, and a line connecting the blocks represents a mapping relationship between records. Any relationship may be expressed by one of two types.

When an element at one level is related to only one element at another level, the mapping is termed simple. The accepted method for the indication of a simple map is a single arrowhead. Figure 6-2(a) indicates the relation of an employee record to his address record. The symbol at the left of the diagram, the single arrowhead, shows that for any given employee there can exist only one address. In many constructions of the schema diagrams, the top of the connecting line does not show any arrowhead. The implication is that even when the arrowhead is missing, it is a simple map. The true name for this relationship is a simple-simple (one-to-one) map but will be called simple for brevity. The right side of the diagram illustrates specific examples of actual records. For example, the employee named Hoffman can only have the address 2134 Adams Street. When the actual records are depicted, the arrowheads cannot be drawn since specific examples of the overall schema are being illustrated.

A relationship where one level has many related elements at another level is known as a complex map. In this case, the double arrowhead is used to indicate a one-to-many map. Figure 6-2(b) exhibits an example of the employee record and its relation to his children's records. Since a given employee can have more than one child, the complex map must be used. Again, the schema is presented on the left side of the diagram while the actual occurrence of the records is shown on the right side. The employee named Hoffman has two children indicated by the two boxes for Tom and Linda.

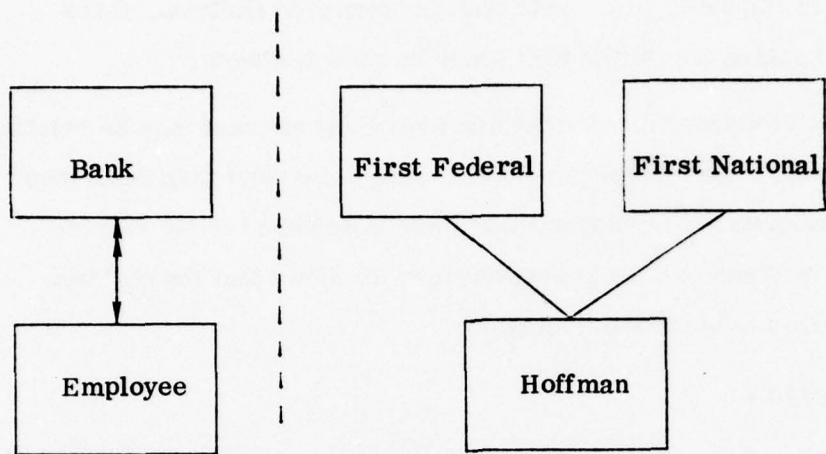


(a) Simple-Simple Map

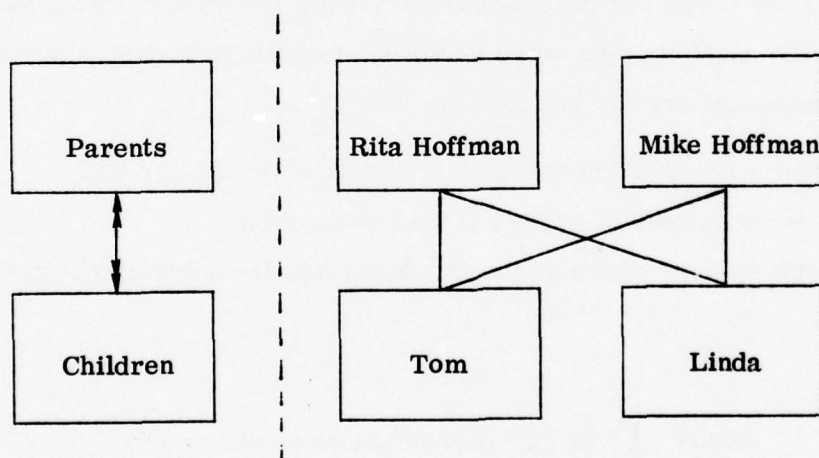


(b) Simple-Complex Map

Figure 6-2. Mapping Relations (1 of 2)



(c) Complex-Simple Map



(d) Complex-Complex Map

Figure 6-2. Mapping Relations (2 of 2)

Situations may exist where many elements are related to only one element, the inverse of a complex map. This mapping is a complex-simple (many-to-one) map and is illustrated in Figure 6-2(c). Note that for employee Hoffman, there may be more than one banking institution with which he does business.

The last variation of maps are relationships where any element may be related to any number of elements. The complex-complex (many-to-many) map describes such a situation. If the spouse of employee Hoffman is considered along with his children, the diagram in Figure 6-2(d) is constructed. It shows that the children Tom and Linda are related to both as is indicated.

6.2.1.2 Bachman Diagrams

An alternate shorthand has been developed to represent specific capabilities of the Integrated Data Store (I-D-S) data base management system (see Chains and Rings (Paragraph 6.3.2)). Due to the integration of the data definition language and the structural relations, a concise definition of this shorthand was developed by Bachman [M].* The basic concepts of blocks** (as illustrated in Figure 6-3 (a)) and inter-connecting lines remain the same, but the usage of the arrows becomes more varied. The arrow points from the master to the detail and may represent different relations. The schema with the blocks and arrows will indicate that:

1. There are some number of records in the system of the master type
2. Each record is the master of a chain of the specified type
3. There are some number of records of the detail type in each such chain.***

*The Bachman arrow " \downarrow " equals " \Downarrow " or " \Uparrow " of previous notation.

**Square blocks are occasionally used for interconnecting blocks.

***Since the definition of a chain is presented in Paragraph 6.3.2, the word relation may be used in place of chain to obtain a similar meaning. The usage of chain in I-D-S structure is related to both the logical relation presented here and the physical relation presented in Paragraph 6.3.

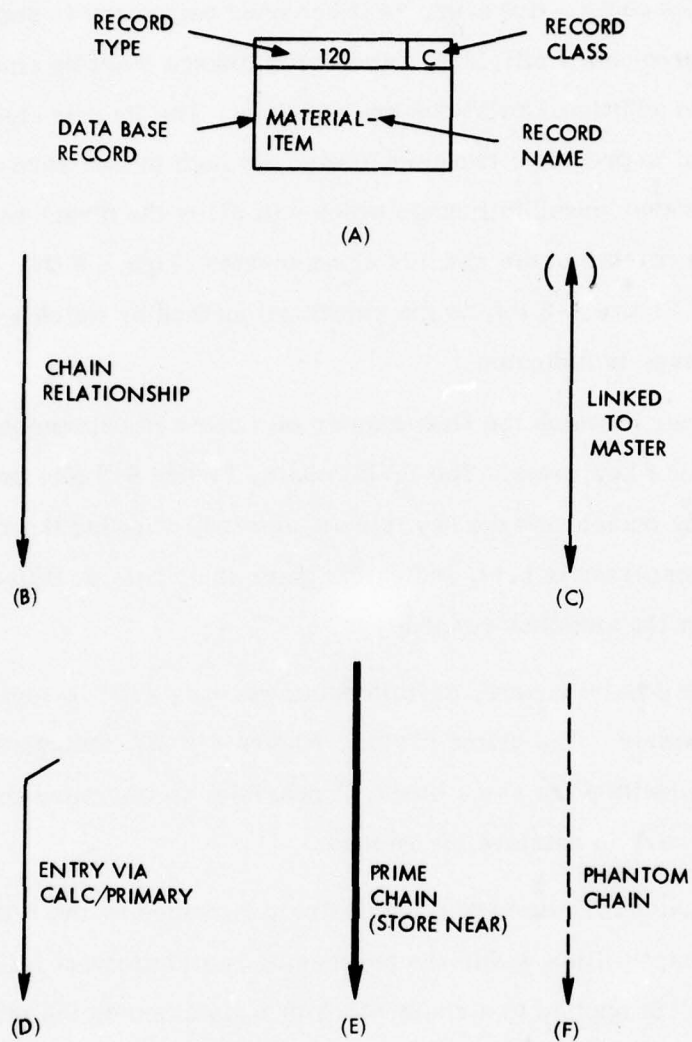


Figure 6-3. I-D-S Symbology

The single arrowheads are used in all I-D-S relationships as indicated by the chain relationship arrow in Figure 6-3(b). The simplest relationship within I-D-S structures have linkage only among details within a chain (Figure 6-21) and not with the master of the chain. However, as it becomes necessary to supply more relations, for increased processing efficiency, additional linkage must be embedded within the structure. Two additional relations are possible. The first is chain prior where the connectors used to provide a two-way linkage, in both the forward and backward directions. Another possible linkage which will allow the direct relation of a given record with the master of the chain is chain master (Figure 6-24). The linked-to-master arrow, Figure 6-3 (c), is the shorthand method by which a chain to master linkage of this type is indicated.

The manner in which the first master of a chain is determined can be varied with the entry of a key value. The CALC chain, Figure 6-3 (d), provides an entry to the master by translating the key into an address, allowing the retrieval of the record. The connector is bent, indicating there is an outside influence describing the entry rather than another record type.

When speed is important, certain detail records must be quickly accessible from a given master. The prime chain, * Figure 6-3 (e), indicates that a record should be placed within the same block, if possible, to minimize the number of disk accesses necessary to retrieve the record.

All the chains discussed previously are maintained by the I-D-S DBMS providing certain capabilities within the processing restrictions of I-D-S. The term "phantom chain" is applied to a chain which is maintained by the programmer to logically express a relation between one record and the next. I-D-S does not recognize "phantom chains", Figure 6-3 (f), and the responsibility for their use is up to the programmer. The only need for "phantom chains" was to bridge "deficiencies" in early versions of I-D-S for structures such as multi-area chains, conditional chains, and temporary chains. A more detailed discussion of chain processing and examples of various chains is given in Paragraph 6.3.2.

*The prime chain is a physical pointer, while the other chains are logical.

6.2.2 Tree Structures

The serial structures described in Chapter 3 cannot adequately depict all the possible logical structures within any given enterprise. A more complex relation must be developed to allow more than one element to follow a given record. Another complicated structure used to define today's data structures is called a tree.

The tree is a structure which takes its name from a similarity with nature's trees. An informal definition of a tree given by Martin [G] states that "a tree can be defined as a hierarchy of nodes with binodal relationships such that:

1. The highest level in the hierarchy has one node called the root.
2. All nodes except the root are related to one and only one node on a higher level than themselves. *

The intuitive notion of a tree becomes obvious without reference to either author's definition. Each of the elements within a tree where a relation may exist is considered a node. The connections (branches) between nodes indicate by what path a relation can be established. The primary node, the node which holds all other nodes together, is the root. Any node may become the root if a subtree is described in which only the relationships from the new root are considered. The number of subtrees contained within a given tree becomes the degree of the tree. Any node which does not have a branch emanating from it becomes a terminal node or leaf, just as the real tree has as its terminal nodes or leaves. The level of any node is defined by

*Knuth [H] presents a more formal definition of a tree which relies on a recursive definition of a tree. Even though the definition is more difficult to understand without investigating the proof, the use of recursion lends itself to trees since they themselves are generally processed in a recursive manner. Knuth's definition is given primarily to allow the reader a choice of definitions. He defines a tree as "a finite set T of one or more nodes so that:

- a. There is one specially designated node called the root of the tree, root (T);
- b. The remaining nodes (excluding the root) are partitioned into $m \geq 0$ disjoint sets T_1, \dots, T_m , and each of these sets in turn is a tree. The trees T_1, \dots, T_m are called the subtrees of the root."

saying that the root is at level 0 and other node levels are numbered one higher with respect to the subtree containing them.*

The basic concepts presented thus far are illustrated in Figure 6-4(a), (b), (c). Considering Figure 6-4(a), the root is node A and contains subtrees defined by nodes {B} and {C, D, E, F}. By picturing the second subtree as a tree, the root becomes C and the subtrees are {D} and {E, F}. Root A is on level zero and D is on level two of the first tree described, whereas C becomes level zero and D becomes level one in the second tree described. The terminal nodes are B, D, F.

The convention used throughout the computing industry turns trees upside down to place the root at the top of the diagram. The tree previously discussed in Figure 6-4(a) becomes the tree in 6-4(b). All properties remain the same.

The relationships between nodes within a tree are compared with a family tree. The root becomes the father of a tree and first level nodes are considered the sons. The relation between multiple nodes within a given tree at the same level are brothers; i.e., the sons of a father are brothers. Examining Figure 6-4(b)**, node A becomes the father and the sons are B and C; B and C are brothers. The father of subtree C is C and the sons are E and D. Extensions to these names have been used such that in subtree C, node A may be the grandfather of D.***

The ordering of the subtrees defined within a tree may be a distinguishing characteristic between separate trees. For example, the trees in Figure 6-4(b) and (c) show the same relationships but differ in the placement of subtree E. By the definition of ordered trees, these two trees are different. However, if this ordering is not considered and only relations between nodes are important, the trees are classified as oriented trees. Thus the trees of (b) and (c), when considered only as oriented trees and not ordered trees, are the same tree. The placement of node E does not make a difference in the oriented tree as long as it comes from node C.

*Martin [G] states the root starts at level one instead of zero, however, for this text

**In depicting trees, arrows are not used (Paragraph 6.2.1.1) since relationships are always from the top down.

***There is no desire to show a prejudice among sexes. The masculine form is used only for conformity with computing literature.

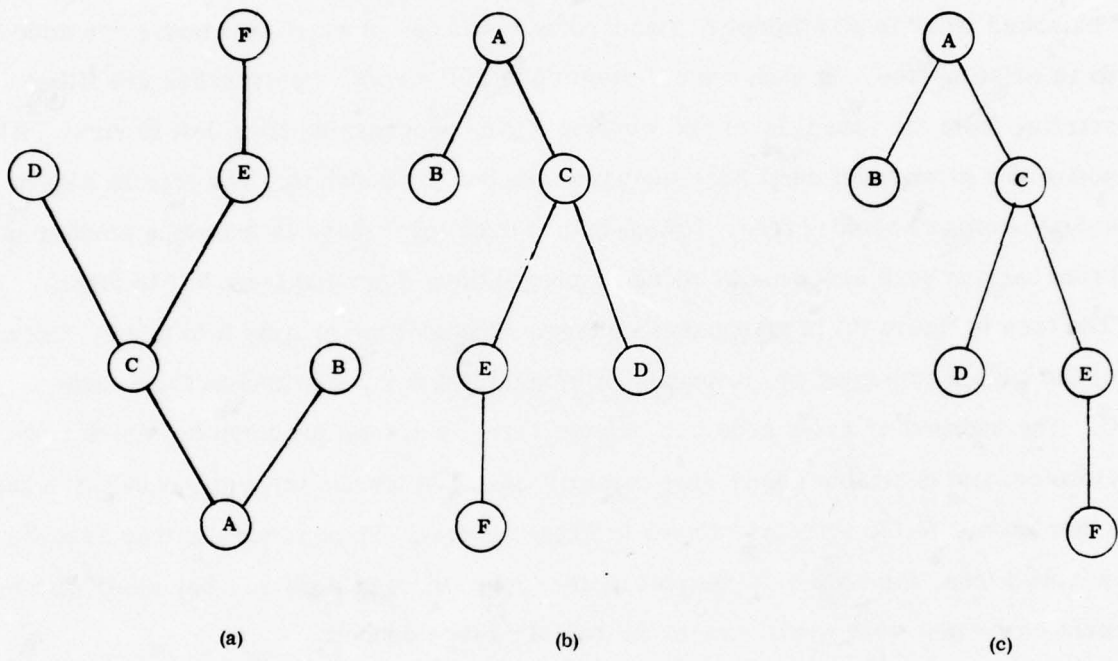


Figure 6-4. Three Trees

A forest is a collection of a series of disjoint trees. If the trees of Figure 6-4 are considered as a whole, then all these trees would comprise a forest.* By converting any node into a root, a forest can be made into a tree and, conversely, the deletion of the root of any tree produces a forest.

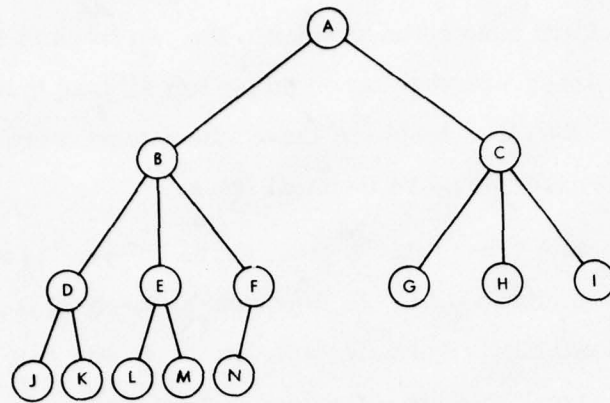
The definition of a tree implies that a node cannot have an ancestor as its descendant. The definition does not specify, however, the number of descendant nodes which can arise from a node. A node may have any number of descendants. When building a tree, the placement of the next node becomes important when ordered trees (as all our trees are implicitly defined) are constructed. The notion of a "balanced tree" is an attempt to standardize the order in which new nodes are added to an existing tree. In such a tree, starting with the root, the branches are filled starting from the lowest level and working down, progressing from left to right. All nodes at a given level must have the same number of branches. The tree in Figure 6-5(a) shows a balanced tree. Notice that at each level there is the same number of branches and each new node is added by progressing down and from left to right. The tree in figure (b) is an unbalanced tree. The addition of node H to node C instead of the node B produced two branches from node B and three branches from node C. The addition of a new node to a balanced tree is always predictable, which provides certain desirable processing capabilities. The tree in (c) is the result of adding a new node O to the structure shown in Figure 6-5(a). To continue the tree as a balanced tree, the node must be placed as a descendant of node F. Any other placement of the new node would violate the balanced tree concept.

When relating trees to the logical relationships between many data structures, the number of descendants may become limited. The definition of a binary tree** provides an entirely new structure with many of the same properties of the trees previously described. The binary tree is a finite tree (possibly empty) in which every node has at most two sons.*** The distinction between ordinary trees and

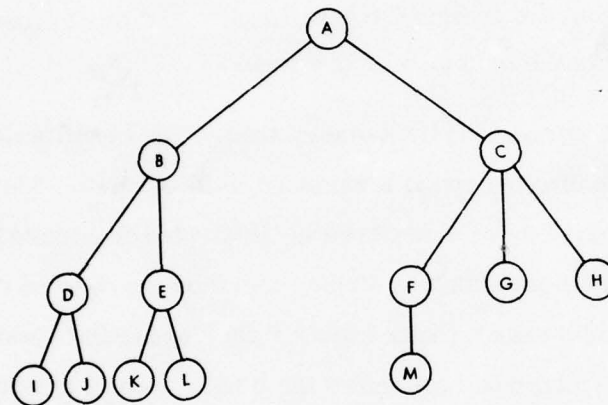
*Some authors use the expression "n-tuply rooted tree" to denote a forest of n trees.

**Similarly a ternary tree (and more generally a t-ary tree for $t \geq 2$) is defined as a root and three disjoint binary trees.

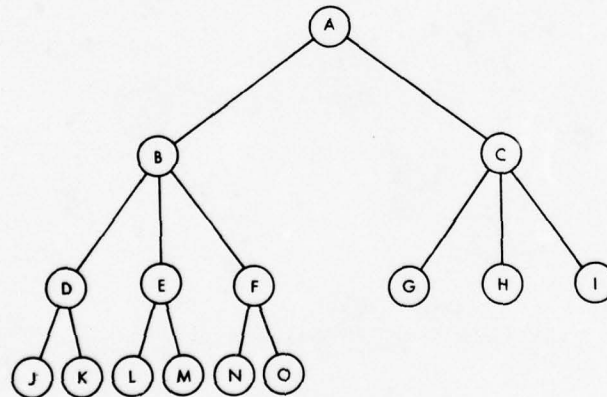
***Knuth [H] defines a binary tree as a "finite set of nodes which is either empty, or consists of a root and two disjoint binary trees called left and right subtrees of the root."



(a)



(b)



(c)

Figure 6-5. Unbalanced Trees

binary trees is important and must be understood. The two trees in Figure 6-6 show two distinct binary trees (the root has an empty left subtree in one case and an empty right subtree in the other case). If these binary trees were to be considered as oriented trees, they would be identical trees.

A special type of binary tree is the "b-tree." The "b-tree" is a tree in which each node has exactly zero or two sons. An important property of the "b-tree" is that there are always an odd number of nodes and that any binary tree with n nodes can be described by a "b-tree" with $2m + 1$ nodes, $m \leq n$.*

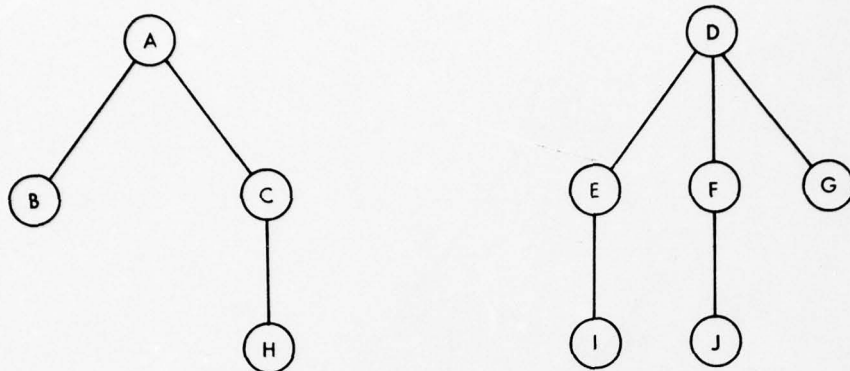
A placement algorithm for the addition of new nodes can be developed for binary trees and "b-trees" just as for ordinary trees. Again, the most commonly used technique for placement is the balanced binary tree.

Any forest can be represented by a binary tree. The benefits derived from the transformation allow specific traversal techniques to be applied. The manner in which a forest is converted into a binary tree is illustrated in Figure 6-7 (a). The corresponding binary tree is obtained by linking together the sons of each family and removing any vertical links except those from a father to his first son. Figure 6-7 (b) shows the resultant binary tree and (c) shows the binary tree in a more natural state. The nodes B, C, and {E, F, G} in (b) can be considered twins.

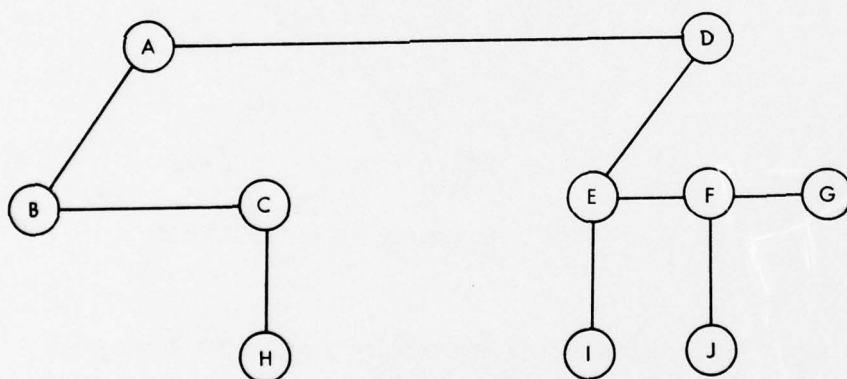


Figure 6-6. Binary Trees

*Such a description would employ "dummy" nodes in the "b-tree" to fill out the binary tree.

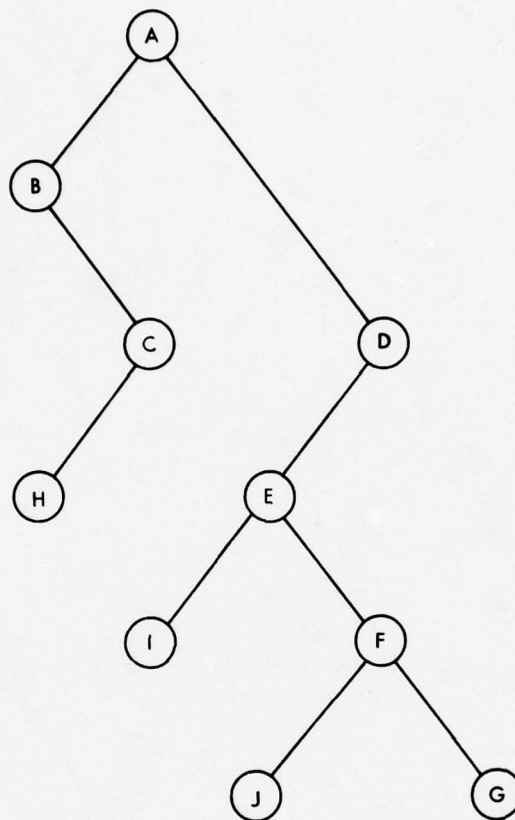


(A) FOREST OF TREES



(B) BINARY TREE REPRESENTATION

Figure 6-7. Binary Representation of Forest (1 of 2)



(C) BINARY TREE REPRESENTATION

NOTE: LEFT-HAND BRANCHES DENOTE SON-RELATIONSHIPS, RIGHT-HAND
BRANCHES BROTHER-RELATIONSHIPS

Figure 6-7. Binary Representation of Forest (2 of 2)

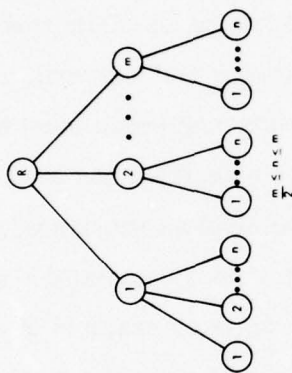
The B-tree, not to be confused with the b-tree described previously, is a specific example of a tree with certain restrictions. Its structure allows trees for large information systems residing on secondary storage to be processed efficiently. The level of the B-tree structure is based upon the number of defining keys and the order of the tree. A B-tree of order m has the following properties:

- i. Every node has $\leq m$ sons.
- ii. Every node, except for the root and leaves, has $\geq m/2$ sons.
- iii. The root has at least 2 sons (unless it is a leaf).
- iv. All leaves appear on the same level, and carry no information.
- v. A nonleaf node with k sons contains $k-1$ keys." [H]

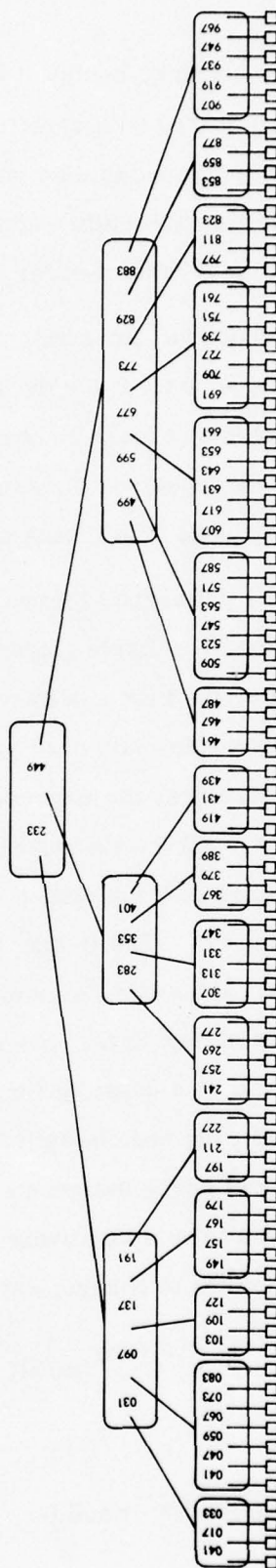
To expand the definition, Figure 6-8 illustrates a theoretical B-tree (Figure 6-8 (a)) and a practical example (Figure 6-8 (b)). Applying the definition to the theoretical diagram, the root must have at least two sons and less than m sons. The example shows a minimum of two sons. The remaining nodes must have at least $m/2$ sons, therefore, the example indicates the sons from one to n , where n is between $m/2$ and m . All leaves must be on the same level as illustrated. Each node contains a key-pointer pair consisting of $k-1$ keys on a node containing k sons (i.e., $k-1$ keys and k pointers). The B-tree in Figure 6-8 (b) is an example of a B-tree of order 7. Note that each node has between three and seven sons and that node A contains five sons and four keys. Insertions and deletions of keys becomes simple where nodes either have a capacity for insertion or when a node is full (i.e., the number of sons = m); the node is split and a key is pushed into the node at a one less level. The number of nodes necessary to be accessed for the worst case can be calculated and shown to be a relatively small value. The level, lv , at which a key exists within a structure of N keys, can be expressed by*:

$$lv \leq 1 + \log_{\lfloor m/2 \rfloor} \left(\frac{N+1}{2} \right)$$

*This formula is derived by [Knuth]



(A) THEORETICAL B-TREE

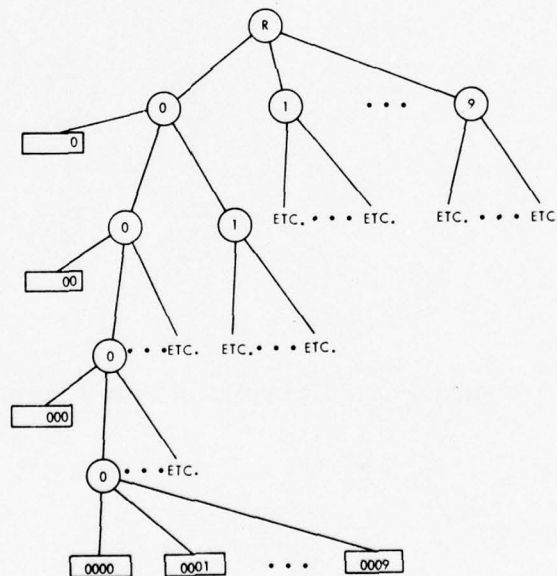


(B) B-TREE OF ORDER 7

Figure 6-8. B-Tree Examples

For a structure containing $N=1,999,998$ and an order of $m=199$ (i.e., up to 199 keys may be stored in any node), the maximum level to define a key is three. This implies that, for a large structure, the maximum depth of the B-tree is relatively small. It can therefore be quickly accessed.

The trie structure (pronounced the same as try) is another variation of the tree structure with restrictions. The n th node contains the n th character or digit representation of the incoming key. For example, if the key can be composed only of digits and only four digits are allowed, the resultant trie structure would be illustrated as in Figure 6-9. The root is an entry into the structure containing branches to all possible first digits of the key. Level one contains the first digit and indicates the existence of the actual data pointer (the box), or a pointer to the next digit of the key. Thus, at level one, the possible valid entries indicate keys of zero with no following digits, zero with some following digits, one with some following digits, etc. The second level from the zero node, indicates branches for a data pointer to value zero-zero, pointer to node zero-one, zero-two, etc. Note that the key zero-zero is a different key than zero.



NOTE: BOX INDICATES ACTUAL DATA RECORD.

Figure 6-9. Four-Digit Trie

6.2.3 Plex Structures (Networks)

Situations will occur where the data relationships cannot be expressed in trees, where there is only one parent for each child. A new structure type has been developed which will allow multiple parents for children. This type structure is called a network or plex structure. In a plex structure, any node may be linked to any other node as in the examples in Figure 6-10. The first example shows each child with two parents, in a conventional family. The last example depicts a situation where the lowest level has four parents.

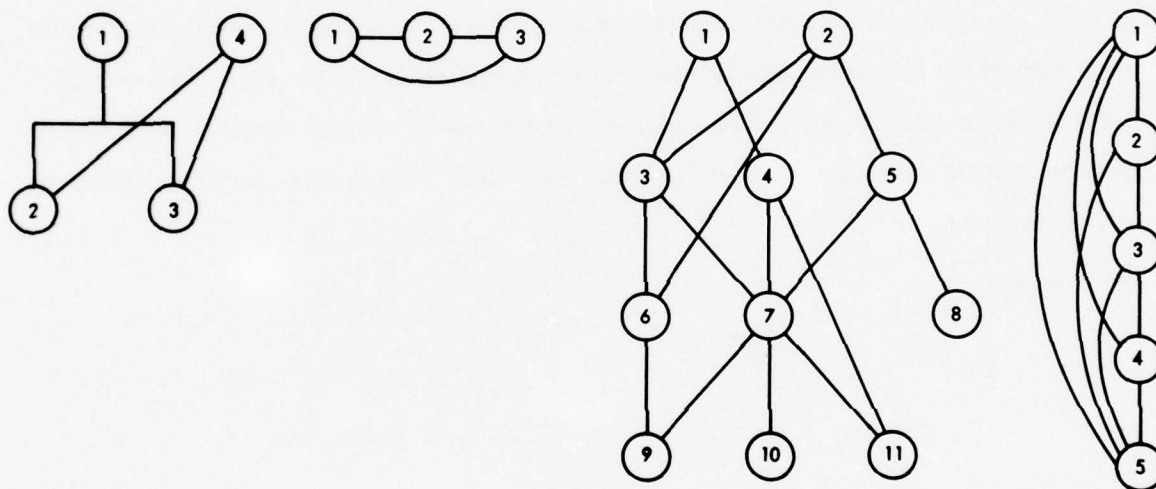


Figure 6-10. Examples of Plex Structures

Many plex structures will depict mappings similar to those for trees; that is, the parent-to-child mapping is complex and the child-to-parent mapping is simple. This is indicated in the diagrams by a single arrow to show a simple relationship and a double arrow to show a complex relationship. Figure 6-11 shows such a combination of structures. The existence of more than one parent for a quotation, (supplier and part) forces this structure to be of a plex type. Notice, however, that there still exists an example of simple mapping where there is only one supplier for each quotation as indicated by a single arrowhead. The example of a complex plex structure, Figure 6-12, utilizes the double arrowheads to indicate that a part can be produced by any supplier, and that any supplier can produce any part. The expansion of the supplier-part relation allows the application programmer to visualize the complete structure.

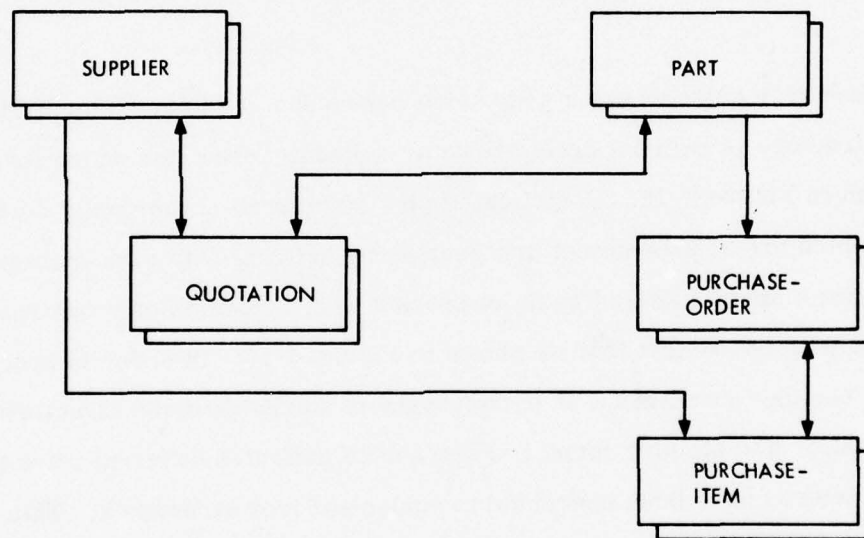


Figure 6-11. Plex Structure of Five Record-Types Used For a Purchasing Application

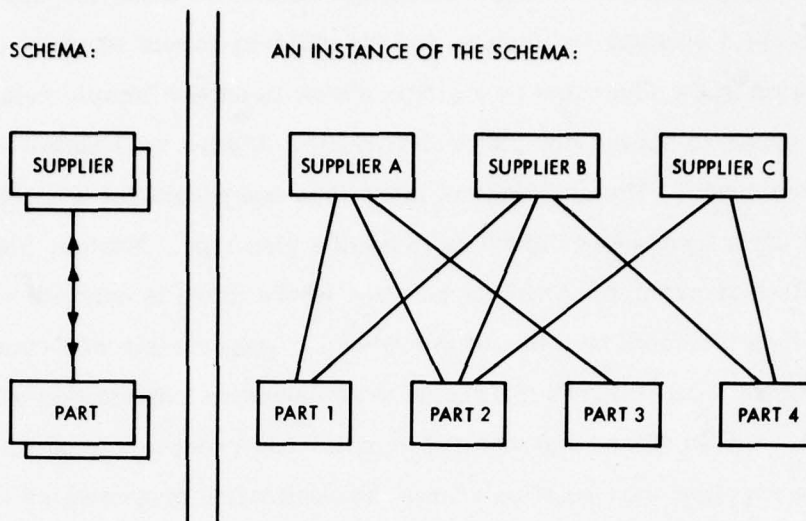


Figure 6-12. Complex Plex Structure With Only Two Record-Types

A situation may arise where a node has a descendant as an ancestor. This type of plex structure is called a cycle shown by a double arrow line which forms a continuous path in Figure 6-13. Complicated plex structures occasionally contain many cycles which are at present not processable by existing data base management systems.* When a special type of cycle is present which contains only one record type, this structure is called a loop as shown in Figure 6-14. In order to process many of these complex structures, it is necessary to subdivide these structures into a series of smaller and simpler forms. Figure 6-15 indicates different ways in which plex structures have been converted to equivalent tree structures. This allows the designer to utilize the data management software at a level where it can handle these structures.

* The I-D-S DBMS is designed specifically to handle plex structures as well as trees and serial structures.

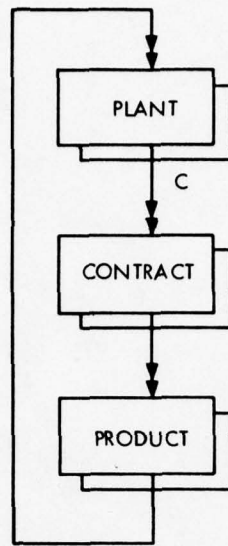


Figure 6-13. Cycle

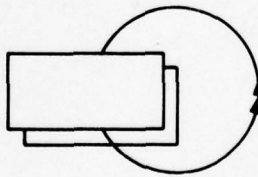
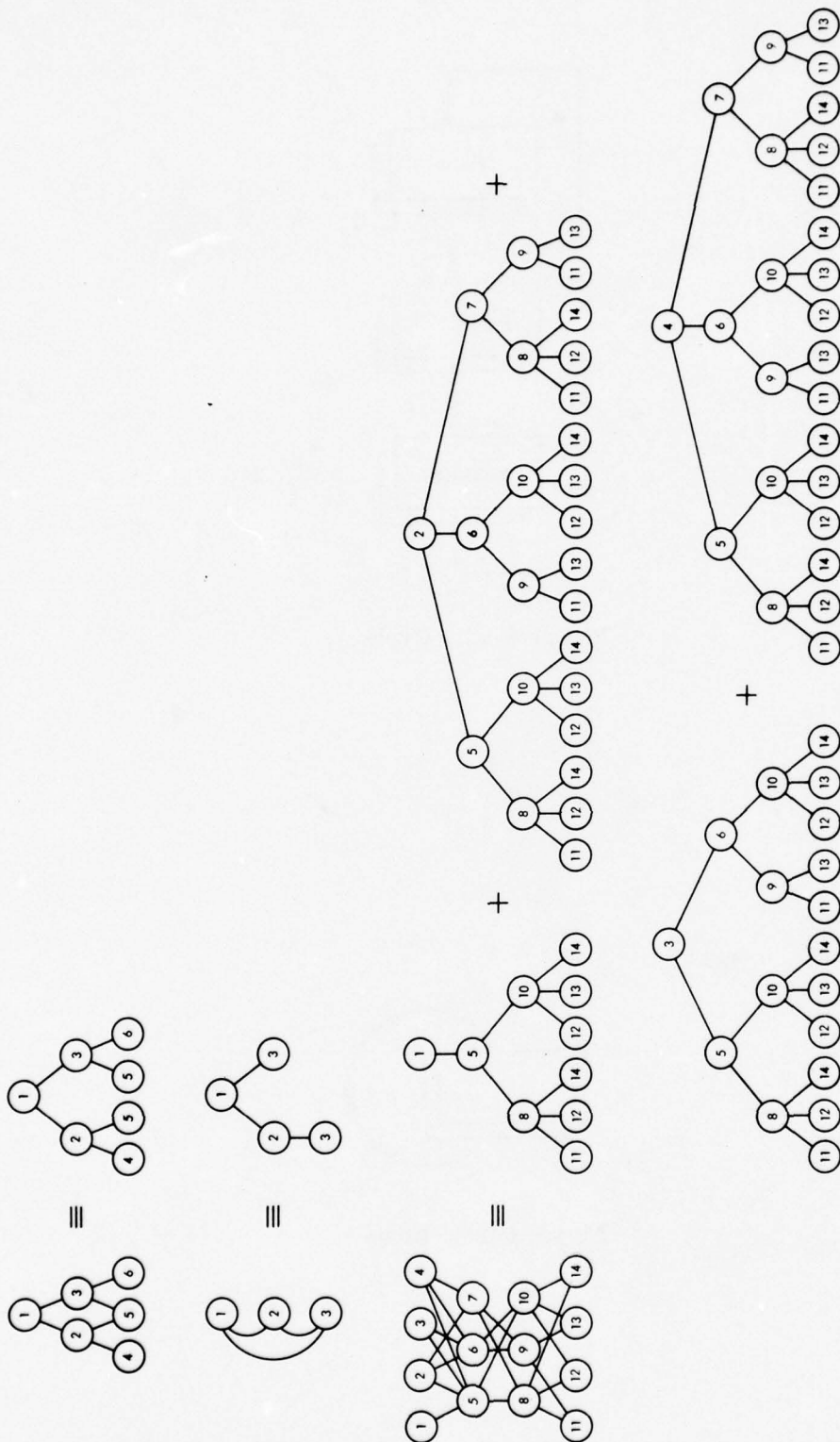


Figure 6-14. Loop



NOTE:
ANY "SIMPLE" PLEX STRUCTURE CAN BE REPRESENTED AS A TREE. OR SET OF TREES, WITH
REDUNDANT ELEMENTS. SOME STRUCTURES CAN TOLERATE MUCH REDUNDANCY, WHILE OTHERS
CANNOT.

Figure 6-15. Equivalent Plex Tree Structures

As previously mentioned, involved plex structures sometimes have to be changed into multiple tree structures. In the example given in Figure 6-12, the plex structure may require a simplification in order to be processed by existing data base management systems. An equivalent set of simpler structures may be created (Figure 6-16) which will still be in accordance with the desired schema. In Figure 6-16, even though there exist two entities for PART, there is no duplication; it is shown for clarification only.

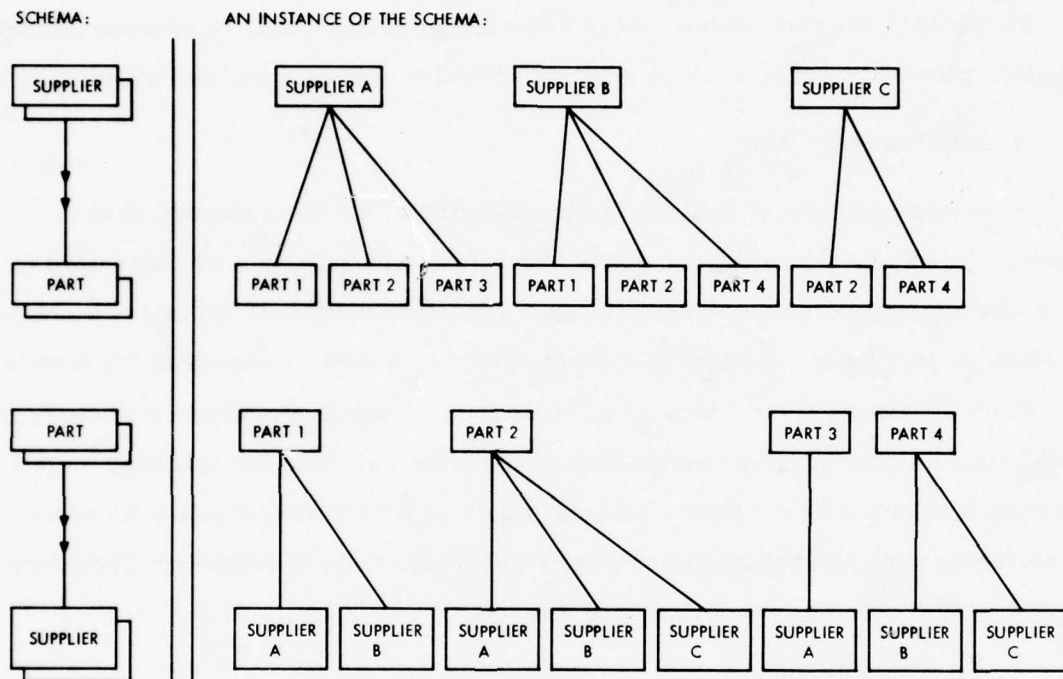


Figure 6-16. Relationship in Figure 6-12 Redrawn as Two Two-Level Tree Relationships

6.3 PHYSICAL STRUCTURES

Once the logical relations in the data base have been established, these relations have to be translated into physical storage. It is imperative that the logical design be examined carefully to match both the design goals (access time, recovery, etc.) and hardware systems capability. The following paragraphs describe the structures involved in achieving these goals.

6.3.1 Pointers

The basic definition of a pointer is a field within a record which indicates where another record is located. The data need not be stored in a physically contiguous area nor in a physically ordered sequence, allowing a much more efficient organization of mass storage. A discussion of pointers for serial sequential structures is contained in Chapter 3.

There are basically three types of addressing methods used to process pointers embedded within a record; machine address, relative address, and record identifier.

6.3.1.1 Machine Address

A machine address is the basic representation of all mass storage data in a pointer. It contains a hardware address, such as a seek address, which is used to locate directly any portion of mass storage. All addressing methods must be eventually translated to this form. In most of today's systems, a data management function is used which translates other addressing forms into the machine address necessary to obtain desired data. Machine addressing is the fastest method for obtaining blocks of information, since it is the most elementary form. Data with this address cannot be moved, since each movement will require calculation of a new address. Therefore, pointers of this type are not device-independent.

6.3.1.2 Relative Address

Relative addressing organizes the records as if they were sequentially numbered within the file, and each pointer is given a unique sequence number, even though data is not in contiguous storage. All noncontiguous spaces are available for other uses and the user is not cognizant of unused space. For the user program to obtain the actual data record, some type of DBMS must translate these sequential numbers into the actual machine addresses.* Since these numbers are in relation to the beginning of the file, and do not refer to the actual placement on disk, the data can be transferred

*Relative addresses may contain multiple addresses within the value of the sequential number. For example, in I-D-S, the relative address contains a page number and a line count.

from one area to another as long as the data base manager can calculate the new addresses. This method of addressing normally takes up less space than machine addressing since the relative address is in a form which is usually less complex. (See Paragraph 4.2.2.1 and Figures 3-3 and 3-4).

6.3.1.3 Record Identifier

This type of pointer is a number or key (e.g., employee number) which is converted into a relative or machine address (See Hashing, Paragraph 7.2.3). Addressing methods of this type are usually specialized for the particular application, and are peculiar to the type of data base being used. Individual records can be moved within a file without modifying all records which point to them.

6.3.1.4 Insertions/Deletions

Consideration must be given to the complexity involved in the deletion of records utilizing pointers. There are two methods which are commonly utilized. The first method reuses existing space for new records and the second deletes records logically. Deleting records logically consists of setting some indicator within the record which will eliminate this record when the file is processed the next time. This periodic restructuring of the file (unload-reload sequence) is necessary to eliminate any logically deleted records. The delays imposed by the unload - reload cycle can become a problem in enterprises which are processed in a real-time environment. (See Paragraph 4.1.2).

Insertion of records may also become time consuming, for the system must follow all pointers to the area where the record is to be logically inserted. An alternate method exists which utilizes pointers but segregates the pointers from the data. The grouping of these pointers into an area is called an index. The index provides advantages, since it occupies little space and, therefore, large portions of the index may be stored in high speed main core storage. Modifications need be made only to the index and not to the data file itself. This provides a high degree of device independence. The index is further discussed in Paragraph 6.3.3.

6.3.2 Chains and Rings

Utilizing the concept of embedded pointers, many data base systems will interconnect records with these pointers to form the logical relationships desired. Whenever a group of records is interconnected, yet physically noncontiguous within a file, it is called a chain (Figure 6-17). The main disadvantage of chained records is that to obtain records in random order, all chained records must be scanned sequentially until the desired record is actually encountered. This contributes to a relatively high overhead which is undesirable in many real-time systems. The file structure will play a large role in the amount of time needed to actually process a chain. For example, if the file is scattered across many cylinders or separate disks, a large number of accesses is necessary to obtain the desired record. However, if the desired record is chained to others within a physical block, the software need only scan this block, thereby providing quick access to the data. High processing time may also be reduced by embedding more chains and by providing more relationships within the structure (i.e., refine access paths through additional structures).

To process a chain, it is necessary to know the first and last data elements. Normally, there is a pointer, referred to as the head, which is the entry point into the chain. The last record will contain some type of flag, referred to as the end of chain indicator, or other mechanisms, so that the user will know when he has processed the entire chain.

There are applications where there are several records containing information pertinent to one particular master record. Efficient processing of these records can be obtained by chaining individual detail records to the master. After processing the master record, all detail records chained thereto will be readily processable. There are many ways to sequence the chain, e.g., FIFO, sorted, ascending, etc., but access to the master must be considered and the sequence optimized for the best storage and retrieval results.

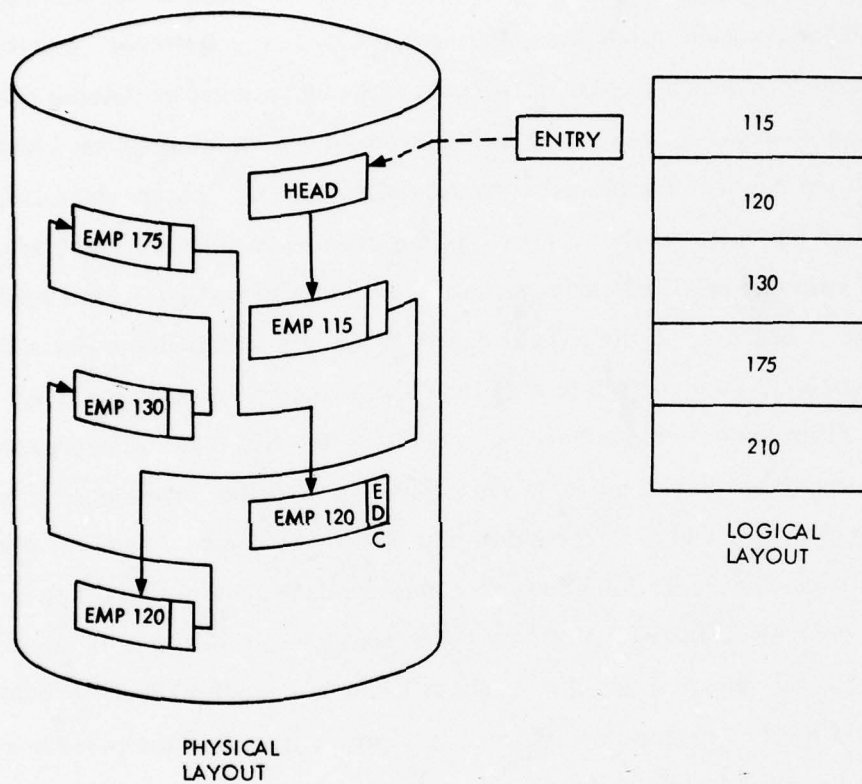


Figure 6-17. Chained Records (Ref. Figure 1-4)

6.3.2.1 Insertions/Deletions

When a new record is to be added in a sorted chain, it is necessary to know the record before the "current" record so that the new record can be inserted in the correct sequence. Additional information, such as indicator bits and added pointers, must be stored within the chain so that the new record can be correctly positioned. There must be some relinking of chains after deletions, which is the same procedure as described for deletion of pointers (Paragraph 6.3.1.4). However, a problem may arise when a record which is part of two separate chains must be deleted and prior pointers are not used. Then, the logical deletion of chains must be used which logically deletes the record of each of the participating chains. Figure 6-18 indicates the need to use logically deleted chains. In the case where chain 1 must be deleted, the physical removal of all records in chain 1, especially record 3, will destroy the lower linkage of chain 2. If the logical deletion of chain 1 is indicated by a bit set in the head of chain 1, then chain 2 is still intact allowing full processing of all records in chain 2. After logical chains have been deleted, the file must be reorganized to remove all unused space. A possible alternative is to utilize two-way chains which indicate both the prior and next record in that particular chain. Two-way chains require twice as much storage for pointers. In large data systems this may be a problem. However, it does allow the reuse of data space which is not possible in one-way chains. In Figure 6-19, the physical deletion of chain 1 can be accomplished, because the intersection of the two chains through record 3 can be traced to the prior record of each chain. As the deletion of chain 1 proceeds, each record is examined for other chains intersecting through the current record. As record 3 indicates, it is part of both chains 1 and 2 and its linkage must be resolved to preserve the linkage of chain 2. The chains pointing to the prior record and next record of chain 2 are reconstructed around record 3, indicating the elimination of record 3 in chain 2. After physical deletion of record 3 is correctly accomplished, chain 2 is still linked. Another alternative available which will eliminate some of the space requirements, is to link the end of chain condition to the head of the chain. This provides a circular type of arrangement

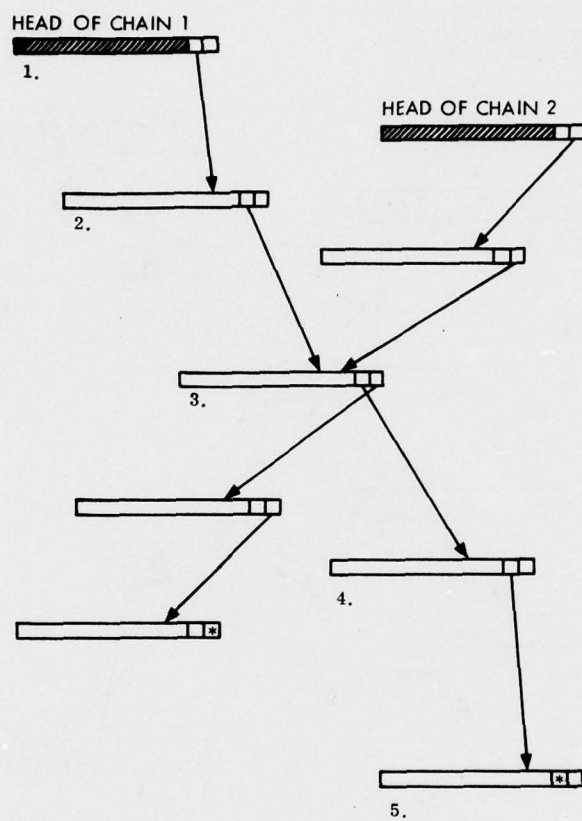


Figure 6-18. Intersecting Chains

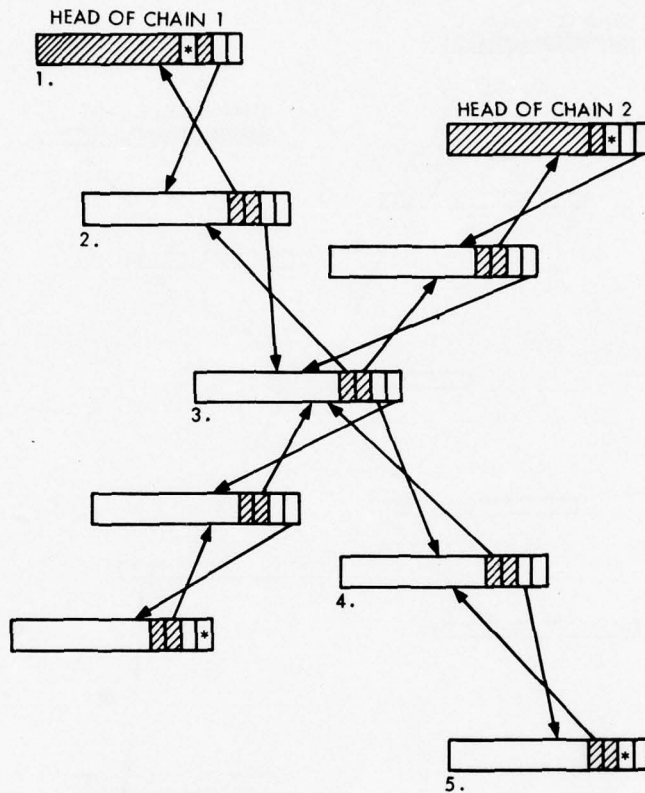


Figure 6-19. Chains With Two-Way Pointers

known as a ring* (Figure 6-20). This will allow chains which intersect with other chains to be physically deleted from the file. A ring can be built with the example given in Figure 6-21. From the point at which the intersection of the chains is found, record 3, the chain 2 linkage can be followed back to the head of chain 2. From this, the prior participant of chain 2 to record 3 can be determined, allowing similar operations to be performed as with two-way chains. The next and prior records in chain 2 are relinked, and the deletion of record 3 from chain 2 is completed.

6.3.2.2 Two-way Chaining Structures

One of the most important aspects of a random structure of chains is the ability to enter at any point in the chain and obtain any other data item within the chain. Normally this is done by starting with the head of the chain and finding the next desired data element. Therefore, it is important to be able to go from any data record to the head of the chain.

Utilizing only a one-way chain, it is time consuming to process all records until the head is found. The two-way chain (Figure 6-22) reduces this time and provides data integrity during system failures. When hardware or programming errors occur and chains are inadvertently broken, portions of data chains may be lost without possible recovery. When only one chain is involved, there is no way to relink these records. However, if two-way chains are utilized, reconstruction of the damaged data base can be accomplished.

Forward and backward pointers (two-way structure) are particularly useful when recovering a broken chain. In a single, forward pointing, ring structure, failure to read a record location will make succeeding records in the chain unavailable.

When reverse pointers exist, the system can start at the head of the chain (Master record), and retrieve the remaining records until the break is found. At this point, the ring is reconstructed around the unreadable locations and processing can continue. The unreadable locations can now be identified and restored from a previous file. Transactions which originally constituted those now missing records can be reprocessed to provide an up-to-date data base.

*The ring is the basic technique used in implementations of Honeywell's I-D-S.

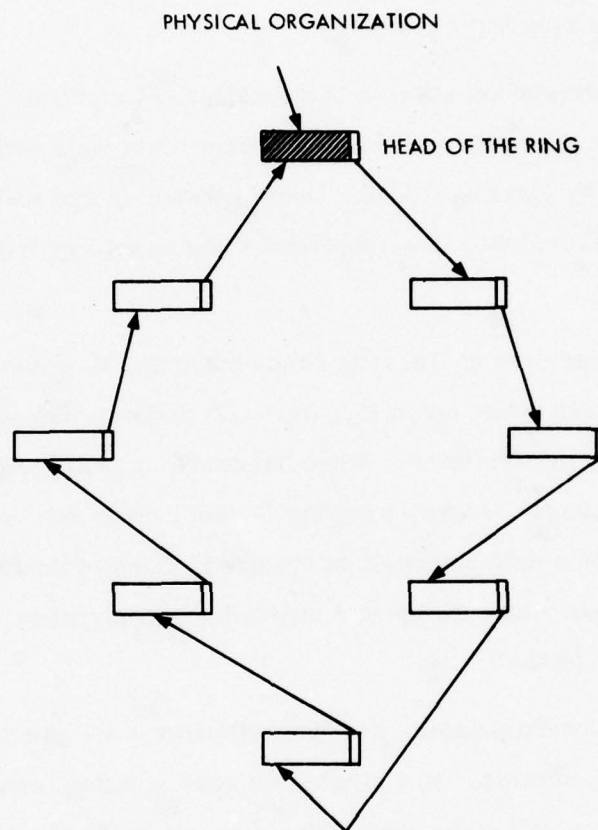


Figure 6-20. Ring With Pointers in One Direction Only

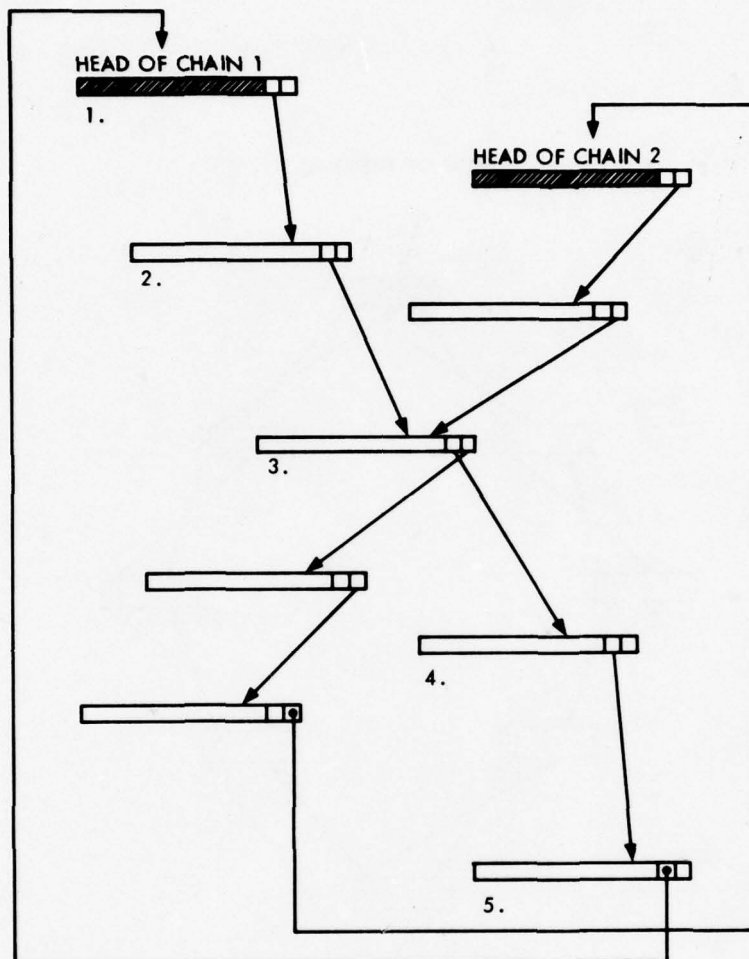


Figure 6-21. Intersecting Rings

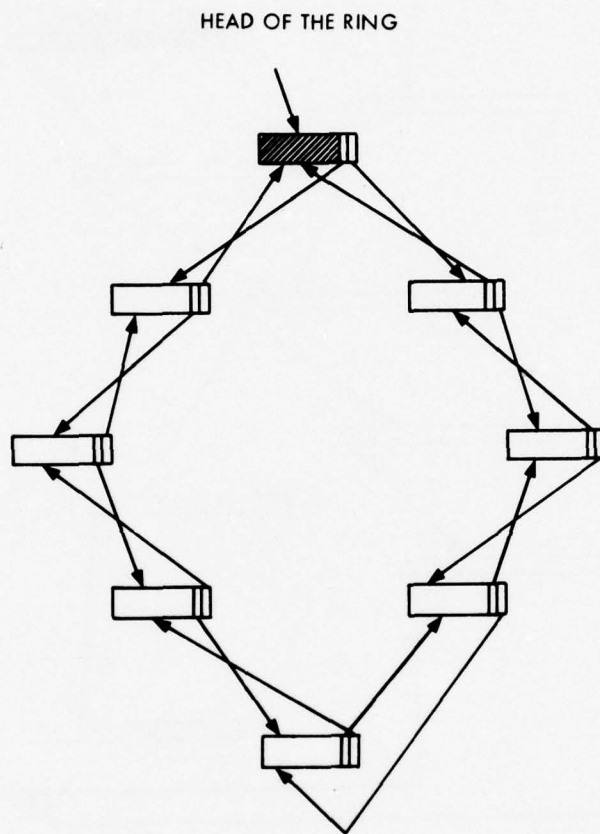


Figure 6-22. Two-Way Chain With Pointers in Both Directions

In the chain or ring structure, a single set of one-directional pointers will provide three basic strategies to add new records. However, each of these is in some respect undesirable.

1. Read all records in a chain sequentially until the appropriate point of logical insertion is found (walk the chain). This is time consuming, since an average of one-half of the records in the data set must be read.
2. Read the entire chain to add a new record to the end of the chain. The first record entered will always be the first one retrieved, i.e., the fastest to find, and the latest entries will always take the longest to retrieve. Thus, additions will always be difficult and retrieval sometimes inefficient. This problem has been resolved in I-D-S by utilizing an end-of-chain pointer in the header record.
3. Enter each addition to a relationship in a position immediately following the head of the chain by means of pointer exchange. In this instance, new additions will be retrieved quickly, and the older records will be pushed away - as in a push down list. Additions will always be efficient, but retrieval can sometimes be inefficient.

6.3.2.3 Multi-directional Chains/Rings

A problem is encountered when direct entry via Index or Randomizing Algorithm is desired to a record at a low level in a hierarchical data base structure. When retrieval of a higher level record is wanted, a single set of pointers will only permit a chain to be walked to its starting point and the record will never be found. For example, direct retrieval of an Employee record, followed by retrieval of the higher level Division record, will necessitate reading all Employee records until the last one which points to the Division record is found, a time consuming operation.

A chain does not alleviate this problem, since the lack of a pointer in the last record of a set prevents a return to the head of the chain.

In Figure 6-23, multi-directional pointer relationships and/or a ring structure may be utilized to overcome many of the preceding problems. In the third alternative, additions are stored counterclockwise immediately following the head of the chain.

The advantages are:

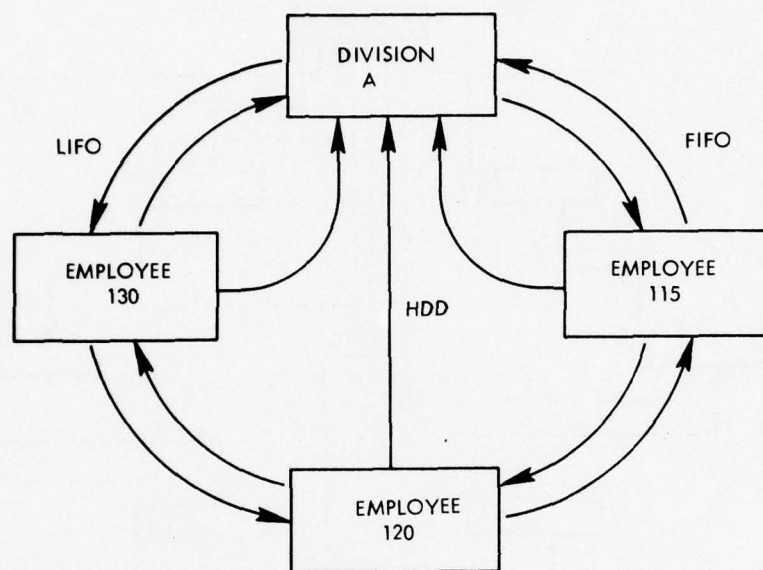
1. The number of accesses which are required can be predicted. When only a single set of pointers is used, four accesses are required to store each new record. These are STORE and RETRIEVE Division records (to enter pointer) and STORE and RETRIEVE Locations for Employee record. The use of backward pointers will require an additional two accesses to reach the next record and enter the backward pointer.
2. Number of accesses is minimized.
3. LIFO processing is supported - i.e., records are retrieved on an LIFO basis. This method makes the most current records available in the shortest possible time.

As shown in Figure 6-23, a second set of pointers, maintained clockwise points from the head of the chain to the first record entered, second entered, etc. As shown, a pointer in Division A contains the address of Employee #115, which, in turn, contains the address of Employee #120. Thus, FIFO processing is facilitated.

There are many types of chain structures which provide different capabilities, some facilitating recovery, others providing accessibility to the head, others permitting fast search time, and combinations of these. Several of these structures are illustrated in Figures 6-24 through 6-29.

6.3.3 Indexing

The preceding examples have all implied that the pointers are embedded within the record itself. There are many instances which require a different method to process these records. One solution is to separate the pointers from the data record and



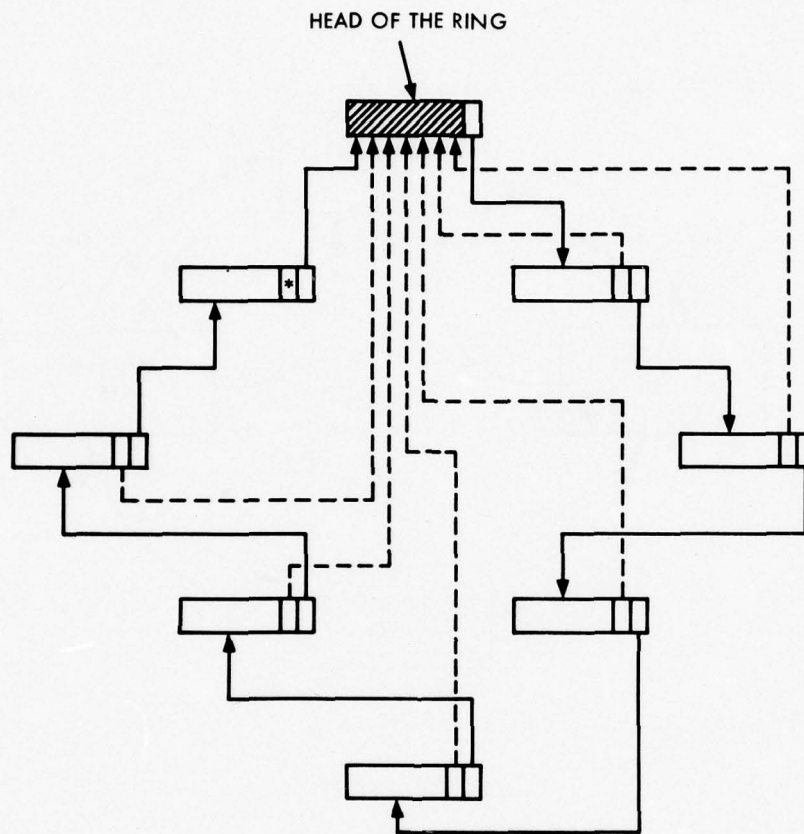
PROCESSING RULES

FIRST : LIFO

LAST : FIFO

HEADED : DIRECT TO MASTER

Figure 6-23. Multidirectional Chains



NOTE:

A RING IN WHICH SECOND POINTER POINTS TO THE HEAD SO THAT A DATA PATH ENTERING THE RING PART WAY AROUND CAN BE ROUTED QUICKLY TO INFORMATION AT THE HEAD.

Figure 6-24. Rings With Head Pointers

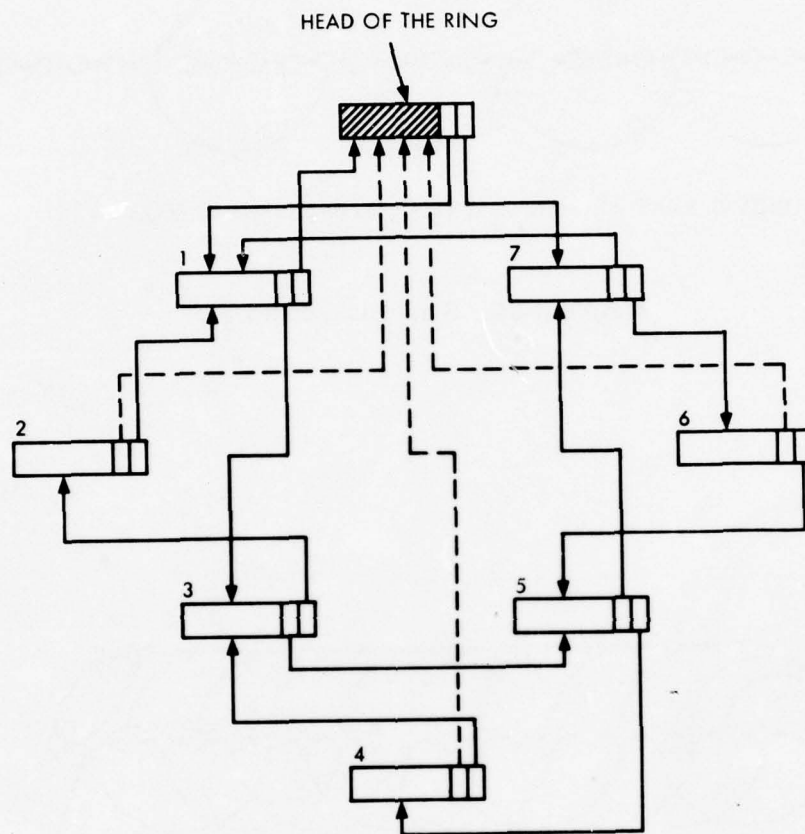
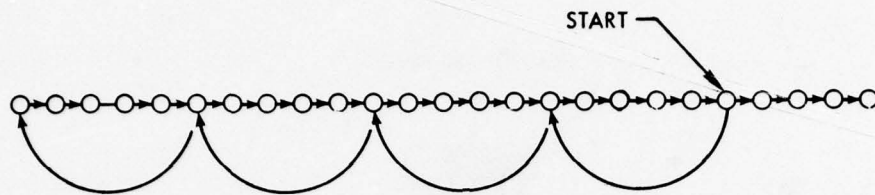
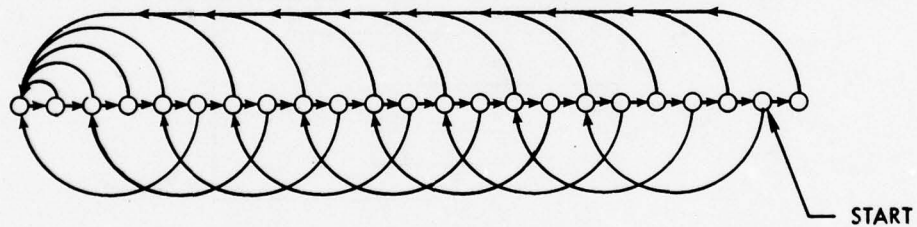


Figure 6-25. Coral Ring



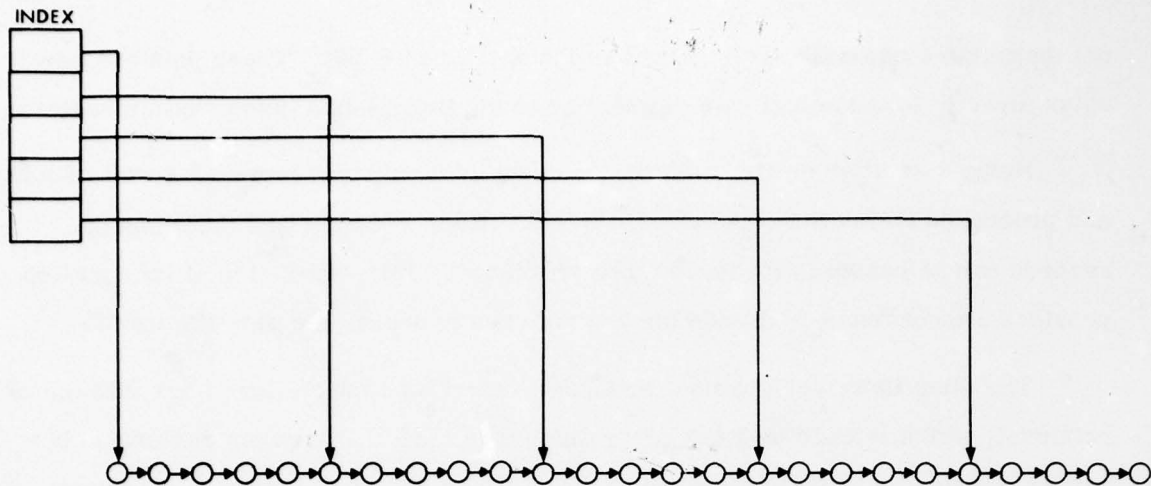
NOTE: PROVIDES FAST SEARCHING OF SEQUENTIAL CHAINED RECORDS

Figure 6-26. Skip-Linked Chain



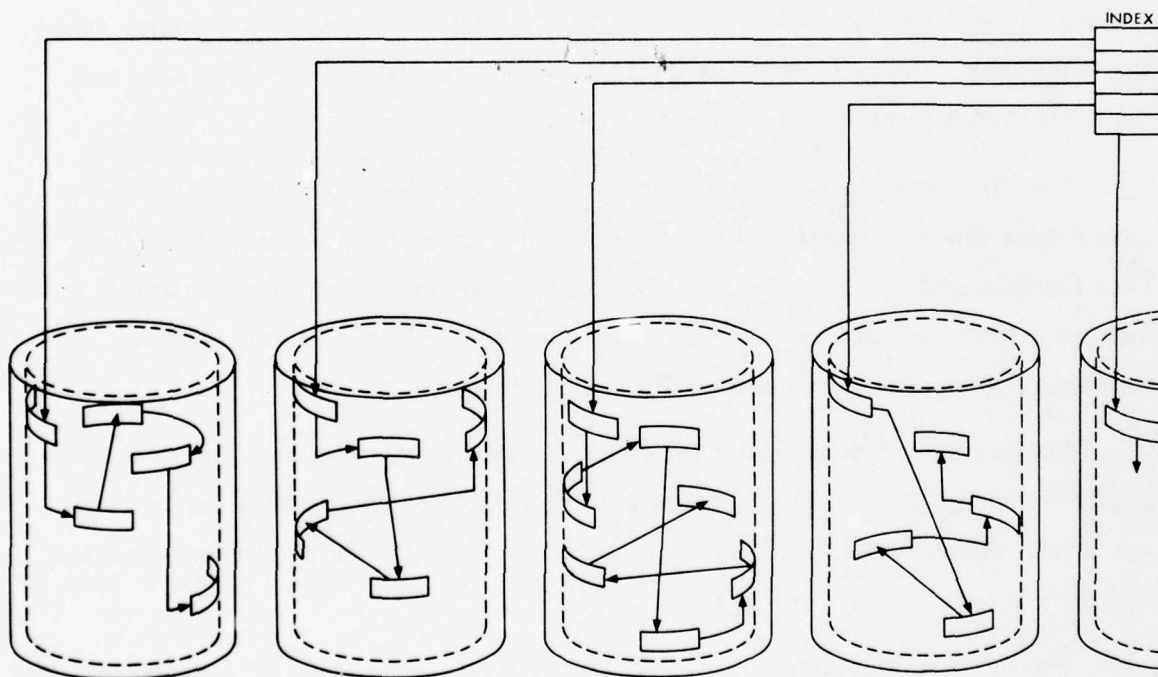
NOTE: SKIP POINTERS ALTERNATE WITH POINTERS TO HEAD OF RING

Figure 6-27. Skip-Linked Coral Ring



NOTE: FOR FAST SEARCH OF SEQUENTIAL CHAINED RECORDS.

Figure 6-28. Multilist Chain



NOTE: ALLOWS SIMULTANEOUS SEARCHING OF SEPARATE CHAIN SEGMENTS.

Figure 6-29. Parallel Cellular Chains

put them into a separate area defined as Index (Figure 6-30). These pointers may refer directly to the actual data element or to the linkage in a given chain structure.

Being a smaller entity, an index can usually be brought into high speed memory and processed faster than the data file itself. Also, additions and deletions of records can be handled with much more efficiency. This separation of information provides a mechanism to decode the key and then to access the data file itself.

The term index designates a method of referring to data, for either storage or retrieval, which is used in many generalized data base management systems. In general terms, an index provides for access either directly to individual records, or to all records of a data set in a defined sequence.

The terms "directory" and "catalog" are often used somewhat interchangeably with indexing to describe the same method of access. Actually, these terms refer to closely related, but nonetheless separate and distinct, methods, which employ similar techniques. Differences are in the relative complexity of the method used and the level of data to which reference is made.

The directory is generally a description of the entire data base and contains information which defines the characteristics, attributes of data items, and data item formats; and describes logical relationships that may exist between data items and records in the data base structure; and provides references to the physical addresses of records to be used in data storage or retrieval.

The catalog is an ordered compilation of data items, providing item descriptions and references to the physical addresses of records to be used in data storage and retrieval. Whereas the directory deals with data analysis, the catalog is concerned with data access.

The index refers to individual data items, the value of each, as access keys, and serves to identify one or more records. The use of an index is sometimes extended to provide retrieval of one or more records on the basis of multiple access keys which represent combined criteria for selection.

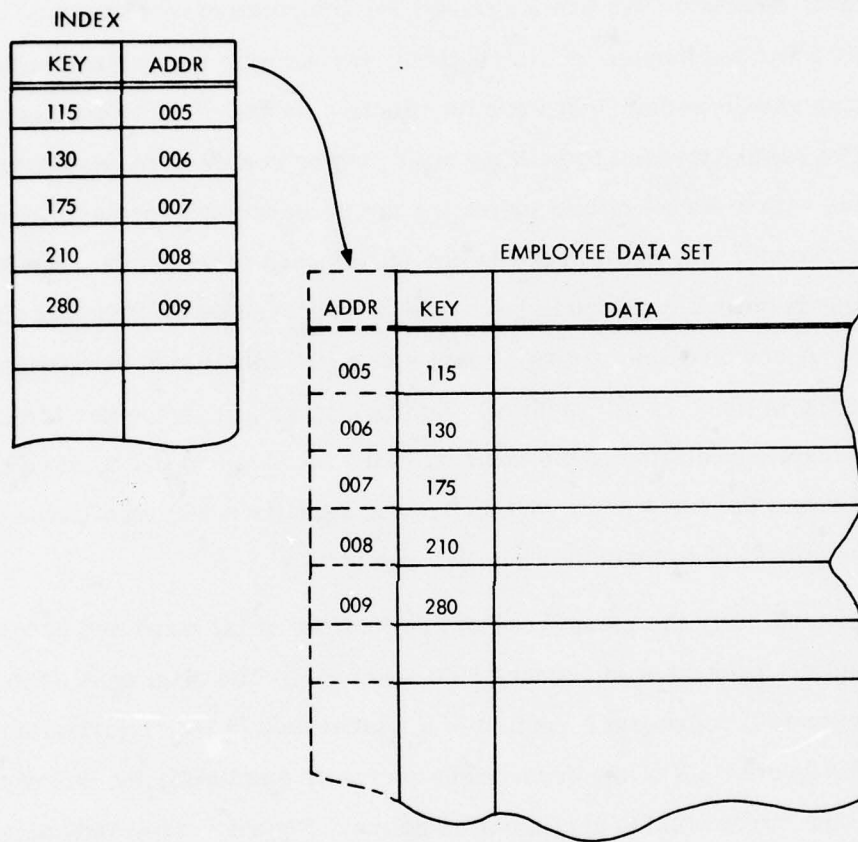


Figure 6-30. Basic Index

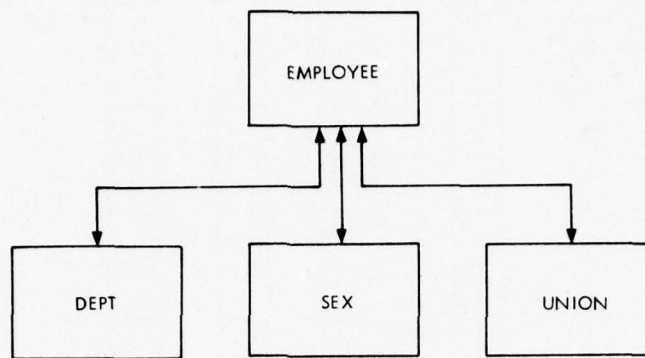
In essence, all three methods of access utilize the relationship between the values of data items and the physical addresses of records containing those data items as a means of locating the desired records. The appropriate key value, in each instance, is paired with the address of the record in which that key value occurs.

6.3.4 Bit Maps

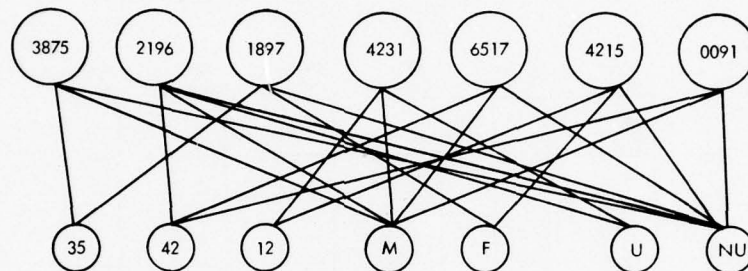
When many separate keys are necessary for information retrieval and these keys have only a limited number of alternatives, two similar specialized indices called a bit map and bit vector/index can be effective means of representing the structure. The basic approach to building a bit map or vector is to associate a given key value with a series of bits indicating the presence or absence of that key value within a record. Normally there is one bit for each record within the file, i.e., for a 1000 record file, a single key could be represented by 1000 bits (28 H6000 words). A considerable saving of data space can sometimes be realized utilizing bit maps in place of full indexes. Address translation becomes trivial, because of a direct correlation of the position in the bit map and the relative position of the record within the file. A bit vector/index associates a series of given key values to a set (block) of records in a similar manner.

To understand fully the many types of applications of bit maps and vectors, a detailed example is presented in Figures 6-31 and 6-32. The diagrams given in Figure 6-31(a) and (b) represent a portion of a typical data base. Information about an employee consists of the department currently employed, the sex of the individual and the current union participation status. Figure 6-31(a) indicates the schema and (b) shows the instance of the schema. A summary of all the attributes of each employee is given in Figure 6-31(c) for a quick reference for this particular example.* The bit vectors in Figures 6-32 (a-d) are examples indicating their construction for the different keys within the employees information. The existence of a one indicates the corresponding record contains an occurrence of that key, while a zero shows the absence of the key. The bit vectors for the sex and union status represent the most efficient usage of bit vectors since there is only a binary choice in any record. Therefore, the sex bit vector, for example, need only represent either

*Figure 6-31(d) represents the same information in the form of a bit map. It is presented only to indicate to the reader an alternate method of representation which is widely used in the references.



(A) SCHEMA



(B) INSTANCE OF SCHEMA

EMPLOYEE	DEPT	SEX	UNION
3875	35	M	NU
3196	42	M	U
1897	35	F	NU
4231	12	M	U
6517	42	M	NU
4215	12	F	U
0091	42	M	NU

(C) SAMPLE DATA BLOCK

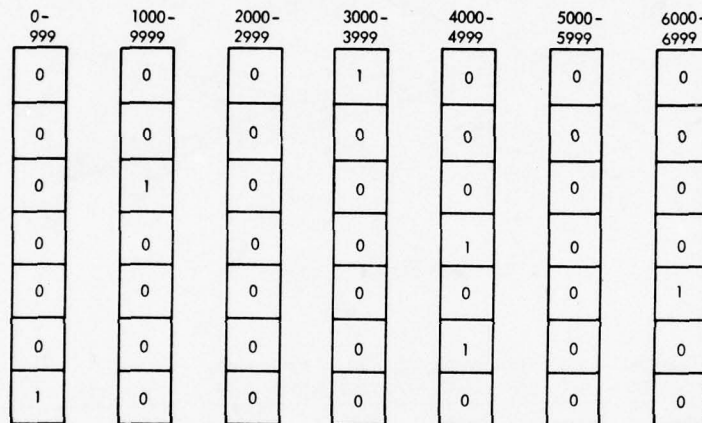
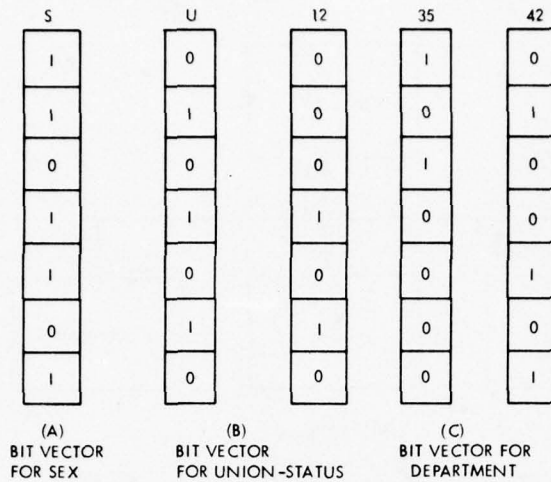
0-999	1000-1999	2000-2999	3000-3999	4000-4999	5000-5999	6000-6999
1	1	0	1	1	0	1

(D) BIT VECTOR INDEX FOR DATA BLOCK

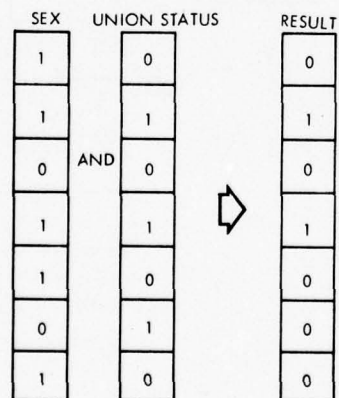
12	35	42	S	U
0	1	0	1	0
0	0	1	1	1
0	1	0	0	0
1	0	0	1	1
0	0	1	1	0
1	0	0	0	1
0	0	1	1	0

(E) BIT MAP FOR DATA BLOCK

Figure 6-31. Bit Map Relations



(D) BIT MAP FOR EMPLOYEE #



(E) RESULT OF AND OF SEX AND UNION-STATUS

Figure 6-32. Bit Vectors

the male or female, since the absence of a male value indicates the presence of the female for that record.

The department key (Figure 6-32(c)) begins to show the weakness of the bit vector strategy since there must be at least one vector for each possible value of the key. Carrying this generalization one step farther, the construction of the employee identification key must be grouped into ranges, otherwise there will be a bit vector for every employee, a considerable waste of space. When the number of records approach a large number, an alternate method becomes necessary, otherwise the bit vectors will require large quantities of space. The bit vectors may correspond to the physical block which contains a logical record. By identifying only the blocks which contain the records with the desired properties, a saving of bit vector space is obtained and a simple scan of the block will find the actual desired records.

The main application of bit vectors is for retrieval of a group of records to satisfy a set of requirements. Multiple requirements can be combined using Boolean algebra* e.g., to find all the males belonging to a union. To find the answer, the bit vectors for sex and union status are ANDed producing the result shown in Figure 6-32(e). This new vector indicates that only the records for employee number 3196 and 4231 will be examined. To find all females in a union status, the sex vector would be EORed with a vector of all ones to reverse the zero/one setting. The resultant complemented vector would then be ANDed to the union vector to identify employee record 4215.

*Boolean algebra truth tables. The row operator and the column operand are combined to produce the resultant table element.

AND	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr></table>		0	1	0	0	0	1	0	1	OR	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>		0	1	0	0	1	1	1	1	EOR	<table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>		0	1	0	0	1	1	1	0
	0	1																														
0	0	0																														
1	0	1																														
	0	1																														
0	0	1																														
1	1	1																														
	0	1																														
0	0	1																														
1	1	0																														

6.4 ADVANTAGES

The main reason that it is important to understand both the logical and physical relationships, whether they be simple or complex, is to relate the structure to the type of data management system in use. Many types of processing techniques are efficient for a tree structure, which, when applied to a plex structure, would be woefully inadequate. Also, different characteristics of the data structure, such as number of levels, number of record types, and master-detail relationships, have an impact on which data management system to select and which techniques to use.

The most important task of any structure is to describe a data base system. If, for example, the data base design requires a simple relationship between parent and child, a tree structure would come to mind. However, if the tree structure cannot satisfy all the relationships, the more complicated plex structure should be considered. A lower level structure can be used to represent a higher level structure if a certain amount of data redundancy is acceptable, e.g., a tree structure can be used to represent a plex structure by replacing certain nodes (Figure 6-15) and a tree structure can be reduced to a serial structure (demonstratable in a COBOL Data Division). Some specific advantages are listed below.

6.4.1 Trees

- Close relation to certain processing techniques.

6.4.2 Plexes (Intersecting chains and rings)

- Able to directly model real life systems
- Can significantly reduce data redundancy
- Can be used to represent serial structures, tree structures, plex structures and many indexing techniques.

6.4.3 Pointers

- Data need not be stored contiguously

6.4.4 Chains and Rings

- Allows variable amount of information
- Separate chains allow efficient processing when only one chain is desired

- More chains provide quicker access and recovery capability

6.4.5 Indexing

- Index can usually be brought into core providing higher processing rates
- Additions and deletions can be handled efficiently without periodic unload/reload
- Data records need not be moved for insertions
- Can provide for both random and sequential processing.

6.4.6 Bit Maps

- May allow more compact indexes, reduces search requirements.

6.5 DISADVANTAGES

The principal disadvantages of the different random structures are presented to help the user of a data base evaluate the desired structure.

6.5.1 Trees

- Limited to a one parent/many child relation
- Long search time to follow all pointers to find desired record
- Recovery may be difficult if a pointer is broken
- Limited capabilities of certain data base software managers to process this structure.

6.5.2 Plexes

- More complex physical structures required
- Recovery may be difficult if pointers are broken within certain types of chain structures
- Some involved plexes can not be processed by the proposed DBTG standard language
- Inability of certain data base software managers to process complex structures
- Additional disadvantages as for chains and rings (Paragraph 6.5.4).

6.5.3 Pointers

- Dependent upon the type of address utilized, a structure may not be device independent
- Algorithms may not allow application independence
- Following pointers may be time consuming in large data base systems
- Insertions/deletions may be complex due to intertwining of record pointers
- Periodic unloads/reloads must be done to eliminate unused space
- Recovery may be difficult if a pointer is lost.

6.5.4 Chains and Rings

- Chains require more space
- Modification of chains may be complex and may require an unload/reload of data base
- Circumstances may require that long chains must be traversed to obtain desired record or additional chains (hierarchical index) must be created to reduce retrieval time.

6.5.5 Indexing

- Not applicable to all types of logical structures.

6.5.6 Bit Maps

- May not be applicable to a wide range of applications.

6.6 APPLICATIONS

Random structures have become the building blocks for most data base systems due to the evolution of modern computing techniques. The need to process information on-line requires a fast access which cannot be provided by the previously described serial structures. The data base designs for existing computers have become so complex as to necessitate use of some of the more complicated structures which have been described.

ISAM (Indexed Sequential Access Method) or VSAM (Virtual Storage Access Method) and Honeywell's ISP (Index Sequential Processor) utilize indexes and pointer strategies to provide efficient data base access. Honeywell's I-D-S (Integrated Data Store) processes rings and chains to provide indexing (structural positioning) as described in some of the examples used in this text.

CHAPTER 7 - RANDOM ACCESS OF RANDOM STRUCTURES

7.1 GENERAL

The random structure is one of the basic types of data organization. A data base may be designed as a purely random structure, or the random structure may be the basis for a more complex data organization.

In a random structure, access of any record is independent of the previously accessed record. Record access is based upon a relationship between the record key and the location of the record in storage.

The indexed sequential file organization, as discussed in Paragraph 5.2.3, is based on a serial index. However, the index may be a random structure and should then be accessed randomly.

7.2 ACCESS METHODOLOGY

Random access of random structures can be accomplished by one of three methods: direct address, dictionary look-up, or hashing. Access, in each case, is based upon the primary key of the desired record.

7.2.1 Direct Address

The direct address method of record access requires that the storage address of a record be known when the record is to be stored or retrieved. The address may be the absolute (machine) address of the record, but is usually a relative address (e.g., page number and line number).*

In the direct address method, the primary key of a record is an address. The key may be an item such as a catalog number, an account number, a portion of an account number, or some reference number assigned to the record. Each key within a file must be unique. Enough storage space must be allocated to the file to accommodate all possible key values. However, if a large number of values remain unused, much storage space will be wasted.

*Not to be confused with the relative address as used in the I-D-S reference code for DIRECT-REFERENCE [V].

7.2.2 Dictionary Look-Up

The dictionary look-up method of accessing a random file depends upon a dictionary of keys and addresses. To access a record, the dictionary, or index, is searched until the desired key is found. The direct address corresponding to the key is used to locate the record. As in the direct address method, the address in the dictionary may be an absolute or a relative address.

The dictionary need not be in any particular order. In such a case, a sequential search must be performed to find any key and the search time may become very high. To decrease the search time, the dictionary may be sorted and then searched using the binary search technique. In any case, the dictionary must be large enough to accommodate all possible key values and the corresponding record addresses. For some applications, the dictionary may be as large as the data file.

7.2.3 Hashing

The most commonly used method of accessing a random file is hashing. With hashing, no dictionary or index is required. A calculation is performed on the key of a record to determine the address of the record in storage.

A key may contain alphabetic and numeric characters. Before the key is transformed into an address, the alphabetic characters must be converted to numeric form.

7.2.3.1 Hashing Techniques

Key to address transformations are accomplished by applying a conversion algorithm to the key. The conversion is an attempt to distribute the records as uniformly as possible throughout the available storage space. If uniformity is not possible, the next objective is to avoid clustering and to minimize record collisions. The algorithm, or technique, chosen will depend upon the set of key values.

The addresses obtained from the transformation may need to be adjusted to fit into the range of available storage locations. The adjustment is a multiplication by the appropriate constant and must be applied to each address resulting from the transformation. As an example, if 500 locations are available for storage and the addresses

calculated contain three digits, the addresses must be multiplied by 0.5 to fall into the 000 to 499 range. The addresses obtained are relative to the beginning of the storage area. (The adjustment process may cause collisions to occur. These will be discussed in Paragraph 8.2.3.2)

7.2.3.1.1 Extraction

One hashing technique is the extraction of bits from a key to form an address. Enough bits must be extracted so that the address contains the correct number of digits for the available storage space. The same bit positions must be extracted from each key of a given file.

Extraction is useful only if the keys are randomly distributed. In practical applications, a truly random distribution rarely occurs.

7.2.3.1.2 Digit Analysis

Digit analysis requires that the distribution of the values of each key position be determined. Those positions having the most skewed distribution are deleted from the key until the number of remaining digits equals the desired address length. For a given file, the same positions must be deleted from each key.

The digit analysis technique may quite often produce erratic results.

7.2.3.1.3 Folding

With folding, a key is partitioned into sections, beginning with the rightmost position. Each section, except possibly the last (leftmost), must have the same number of digits as the address. The key is then folded, similar to folding paper, and digits folded into the same position are added together. (See Figure 7-1 for an illustration of folding.)

7.2.3.1.4 Shifting

The shifting technique partitions a key as for folding. The sections are then shifted so that the rightmost positions of the sections are aligned. The shifted digits are added together to obtain the address. (See Figure 7-2 for an illustration of shifting.)

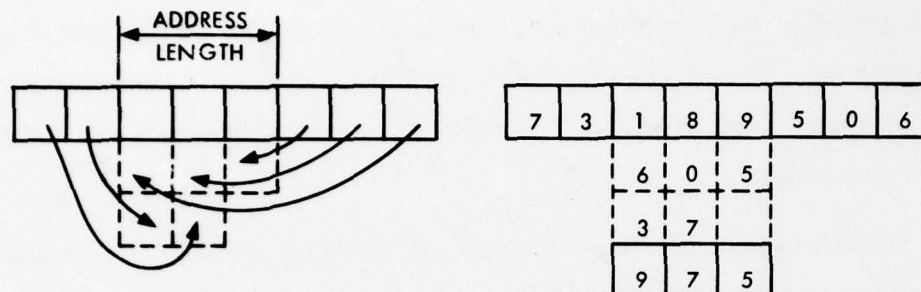


Figure 7-1. Folding

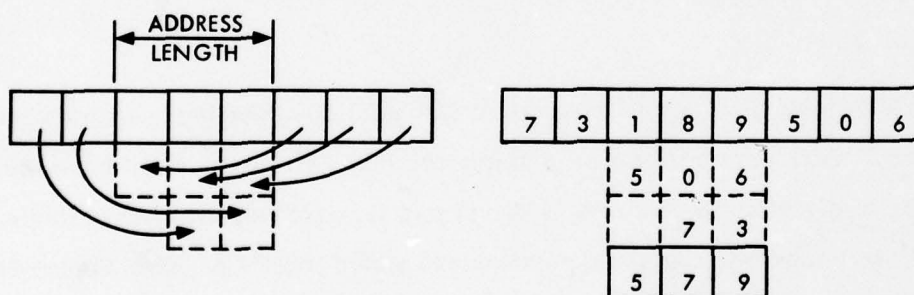


Figure 7-2. Shifting

The folding and shifting techniques illustrated may be modified. Digits, which have not been folded or shifted, may also be used to obtain the address. Another variation is to ignore all carrying in the addition process. Whatever method is chosen, it must be used consistently on any one file.

7.2.3.1.5 Mid-square Method

The mid-square method is a hashing technique that gives fairly good results. With this method, a key is multiplied by itself. The product is truncated at both ends until the number of digits remaining is equal to the appropriate address length. For example, the key 4263 multiplied by itself yields 18173169. If the address length is three digits, the desired address is 731 or 173. Whichever three digits are chosen, the same positions must be used for all keys of any one file.

7.2.3.1.6 Division

Of all hashing techniques, the division method consistently gives favorable results. In division, the key is divided by a number and the remainder is used as the relative record address. The divisor chosen should be close to the available number of storage locations and should be either a prime number or a number with no small factors. Such a choice of divisor tends to minimize the occurrence of collisions.

As an example of the division method, if 500 storage locations are available, the divisor might be 499. The key, 102691, divided by 499 gives a remainder of 396 for the record address.

7.2.3.1.7 Algebraic Coding

The algebraic coding technique is based upon division. Each digit of a key is considered to be a polynomial coefficient. The polynomial obtained is divided by another polynomial and the coefficients of the remainder give the record address. The degree of the divisor must be chosen so that the number of digits in the remainder will be appropriate for the number of storage locations available.

To use algebraic coding, the key, 102691, is considered to be $x^5 + 2x^3 + 6x^2 + 9x + 1$. After dividing by the chosen polynomial divisor, the coefficients of the remainder determine the record address. For any given file, the same divisor must be used for each key.

The algebraic coding technique often provides satisfactory results, but the choice of a polynomial divisor is difficult.

7.2.3.1.8 User-Defined Function

If all of the documented key to address transformations prove unsatisfactory for a particular application, the user may define his own hash function. The primary objective of such a function is to distribute the keys uniformly across the available storage space. The number of collisions will be minimized.

The hash function should also be a fast calculation. Excessive time spent in computing addresses decreases the benefits of hashing.

7.2.3.2 Overflow

When two keys hash to the same address, a collision occurs. The two keys are called synonyms and result in a condition known as overflow. On record storage, if the address calculated for a particular record is already occupied, an overflow occurs. Since every hash function will usually produce some collisions, provisions must be made to accommodate the overflow records.

7.2.3.2.1 Chains

Overflow records can be incorporated into a data base by chaining. A chain, as described in Paragraph 6.3.2, is a linked list in which each record contains a pointer to the next record in the list. In overflow chaining, a chain consists of those records whose keys hash to the same address. The sequence of the records is simply the order in which the records were stored. To access a particular record, the chain must be traversed until the record with the desired key is found. If the key is not found after traversing the entire chain, the record does not exist.

Only the first record of a chain will be stored at the address determined by the hash function. This record will reside in the primary storage area of the file. In addition, an overflow area could be allocated to the file and the remaining records of the chain stored in the overflow area.

7.2.3.2.2 Buckets

The need for chaining records can be eliminated by dividing the available storage space into small sections called buckets. Each record is stored in one of the buckets and synonyms are stored in the same bucket.

The use of buckets will decrease the number of overflow records, but some overflows will occur. The number of overflows depends upon the hashing function and the size of the buckets. A small bucket size may cause a large number of overflows. A large bucket size will decrease the number of overflows but increase the search time. When the bucket address of a particular record is calculated by the key to address transformation, that bucket must be searched until the desired record is found.

The packing density of the data also affects the number of overflows. Packing density is a ratio of the number of records stored in the buckets to the capacity of the buckets. If the packing density is high, approaching 100 percent, when a data base is created, records added to the data base are likely to overflow. A lower packing density, perhaps 80 percent, will more easily accommodate additional records. However, if the packing density is too low, e. g. less than 50 percent, storage space may be wasted. The packing density selected should be influenced by the expected activity involving the data base, particularly the volume of record additions.

7.2.3.2.3 Overflow Storage

The most careful selection of hashing function, bucket size and packing density will not entirely eliminate the occurrence of overflows. Therefore, a data base management system which utilizes hashing for random access must provide some means for handling overflow records.

7.2.3.2.3.1 Open Addressing

If the bucket in which a particular record is to be stored has already been filled, the record may be stored in another bucket in the primary storage area. A method of accomplishing this is open addressing, sometimes called the consecutive spill method. With open addressing, an overflow record is stored in the next consecutive bucket which has space available. The advantage of this method is that an overflow record is usually stored near its home bucket, the bucket determined by the hashing function. However, if the bucket size is small, many buckets may have to be searched to locate a particular record. In addition, a bucket may become filled with overflow records and would not be able to accommodate those records assigned to it as their home bucket.

A slight variation of the consecutive spill method is to skip a given number of buckets in the search for available space. Storage space is considered to be circular and the number of buckets to be skipped should be chosen so that the entire storage space may be searched if necessary.

7.2.3.2.3.2 Free-Space Directory

A free-space directory indicates which buckets have storage space available. When an overflow record is to be stored in the primary storage area, the directory can be accessed to determine the bucket in which the record is to be stored. A pointer to the record must be inserted in the record's home bucket. The use of a free-space directory eliminates the need to search buckets for available space. However, space is required for the directory and for the pointers. Space used for pointers in a bucket decreases the space available for record storage in that bucket.

7.2.3.2.3.3 Separate Overflow Area

Rather than store overflow records in the primary storage area, a separate overflow area can be used to store these records. The overflow area may reside on the same cylinder as the primary storage area or it may be an independent area on another cylinder.

Records in an overflow area may be handled in several ways. Individual records can be chained, as described in Paragraph 7.2.3.2.1. Each bucket in the primary area that overflows must contain a pointer to the start of its overflow chain. The chaining method will perform satisfactorily unless a large number of overflows occur. When the chains become long, search time is increased.

To reduce the search time involved in traversing chains, the overflow area may be divided into buckets. When a bucket in the primary area overflows, the record is placed in an overflow bucket and a pointer to the overflow bucket is placed in the home bucket. Additional overflow records are stored in the same bucket. If an overflow bucket itself overflows, records are then stored in the next overflow bucket. The home bucket must contain a pointer to the overflow bucket containing its overflow records. An overflow bucket may also require a pointer. If all of the overflow records from a particular home bucket cannot be stored in the same overflow bucket, the overflow bucket must point to another bucket in the overflow area which contains the additional records. Thus, the buckets form chains, but the chains are considerably shorter than those of individual records.

Other methods which may be used to store records in an overflow area are open addressing and the free-space directory method. The descriptions given in Paragraphs 7.2.3.2.3.1 and 7.2.3.2.3.2 refer to the primary storage area, but the methods can also be used for buckets residing in an overflow area.

7.2.3.2.3.4 Distributed Overflow Space

Overflow buckets may be distributed among the home buckets in the primary storage area. If the home bucket calculated for a particular record has already been filled, the record is stored in the next consecutive overflow bucket. When an overflow bucket becomes full, additional records are stored in the next consecutive overflow bucket. This method is similar to distributed free space in Paragraph 5.2.3.4.2. The advantage of distributed overflow space is that an overflow record is usually stored close to its home bucket.

7.2.4 Indexed Sequential Access

If the index to an indexed sequential file is a random structure, the index should be accessed in a random manner. Although random structures can be accessed sequentially by scanning, this technique provides no practical benefits.

7.2.4.1 Random Index

A random index can be accessed by using any of the hashing techniques described in Paragraph 7.2.3.1. Hashing is used to access only the index. The desired record is found by obtaining, from the index, the location of the block containing the record. As in all cases of indexed sequential organization, the designated block is scanned to find the appropriate record. This method has the advantage of applying the hashing technique to a uniform set of small records, simplifying the packing density problem.

7.2.4.2 Index Overflow

The topics discussed in Paragraph 7.2.3.2 pertain to the overflow of records in a file. However, the methods used to handle overflow records can also be used to handle overflow keys in a random index.

7.3 INSERTIONS/DELETIONS

The insertion and deletion of records in a random file is a simple matter. To insert a record, its storage location is found by using one of the hashing techniques described in Paragraph 7.2.3.1. The same technique must be used for all records of any one file. If the storage address or bucket address calculated has already been filled, the new record must be stored in an overflow area. Any of the methods for handling overflow, described in Paragraph 7.2.3.2.3, may be used. Again, the same method must be used for all records of any one file.

The deletion of a record from a random file can be accomplished by marking the record as deleted, without actually removing it from the file. Future searches

of the file can then ignore the record. A new record can overlay the deleted record if its key calculates to the same address and the record is of the appropriate size.

7.4 ACCESS CONSIDERATIONS

When a random file is created, the records which will be accessed most frequently should be stored first. Thus, they will occupy the home buckets and will have the lowest access time. Less frequently accessed records will be stored in overflow buckets when their home buckets are filled.

While a file is being used, the DBMS should record statistics regarding the access of records. Over a period of time, the access pattern may change; that is, records which were seldom accessed when the file was created may now be accessed frequently. If these records reside in an overflow area, it would be advantageous to reorganize and reload the file, placing the most frequently accessed records in the primary storage area (or home buckets).

File reorganization is an expensive and time consuming operation. It should only be performed when the access pattern has changed drastically. Reorganization, to be worthwhile, must result in greatly improved performance of the DBMS.

7.5 ADVANTAGES

The three methods of randomly accessing random structures, presented in Paragraph 7.2, include direct address, dictionary look-up and hashing. Of these, the direct address method provides the fastest access because a record's key specifies the storage location of that record. Both the direct address and dictionary look-up methods yield unique record addresses, but dictionary look-up requires less record storage space.

Hashing appears to be the most favorable random access method. With hashing, no space is required for a dictionary or index. Additionally, no time is consumed in searching for keys.

7.6 DISADVANTAGES

The direct address method of random access is most wasteful of storage space. The dictionary look-up method may require a large amount of space for its dictionary and may also consume a large amount of search time in finding keys.

The main disadvantage of hashing is the overflow that usually occurs. However, several techniques, presented in Paragraph 7.2.3.2.3, are available for accommodating overflow records. The file may require periodic reorganization to increase the efficiency of record access.

7.7 VARIATIONS

The inverted list file organization (Paragraph 5.2.4) contains an index composed of keys and pointers. The pointers may be primary keys of randomly stored records. In such a case, any one of the hashing techniques in Paragraph 7.2.3.1 may be used to determine the storage location, or bucket address, of a record.

In reference to the I-D-S chain in Figure 4-7, CHAIN-ORDER STORED WITHIN TYPE, additional pointers could be embedded in the records to link all records of a particular type. The pointers are illustrated as broken lines in Figure 7-3. The resulting structure allows all type n ($n = 1, 2, 3, 4$) records to be found by following pointers, rather than traversing the entire chain. The additional pointers form a chain for each record type. If a pointer is inserted into the last record of a particular type, pointing to the head of that "type" chain, a ring structure results (See Paragraph 6.3.2.1).

Knotted lists are chains in which each element of the list contains a direct link back to the head of the list. In software systems, such as I-D-S, a given record can be linked into more than one chain. Furthermore, each record can have a direct pointer back to the head of each chain into which it is linked. Advantage can be taken of this capability to model multifaceted relationships such as is evidenced in multi-level inverted indices.

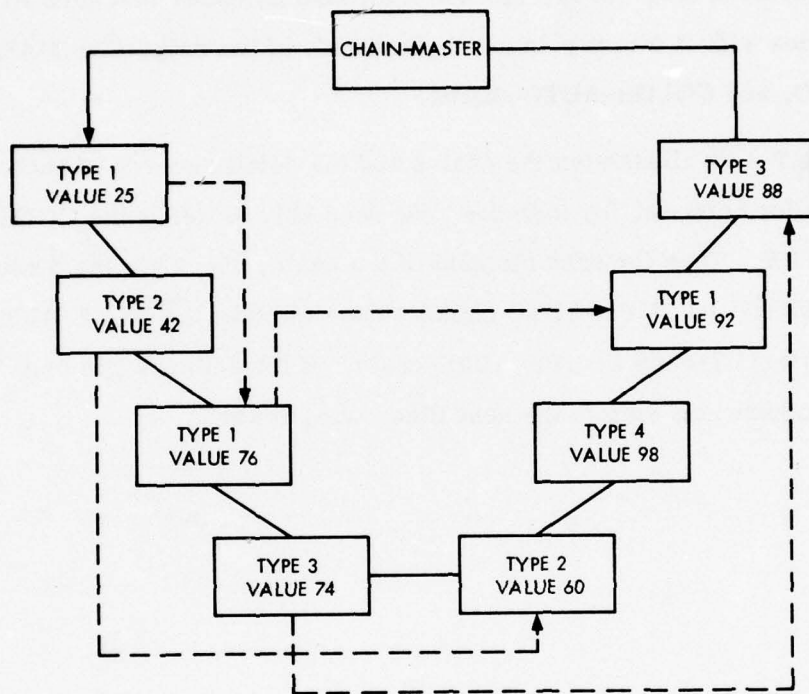
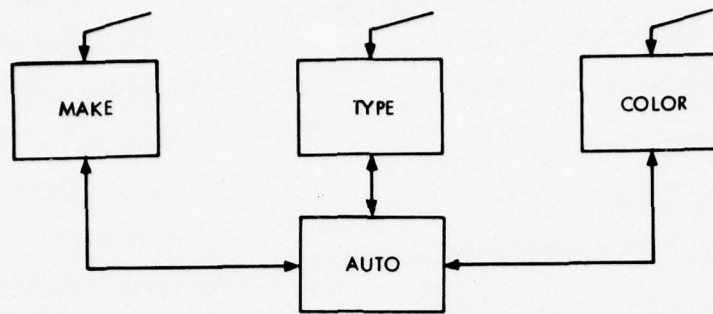


Figure 7-3. Chain-Order Sorted Within Type,
With Additional Pointers

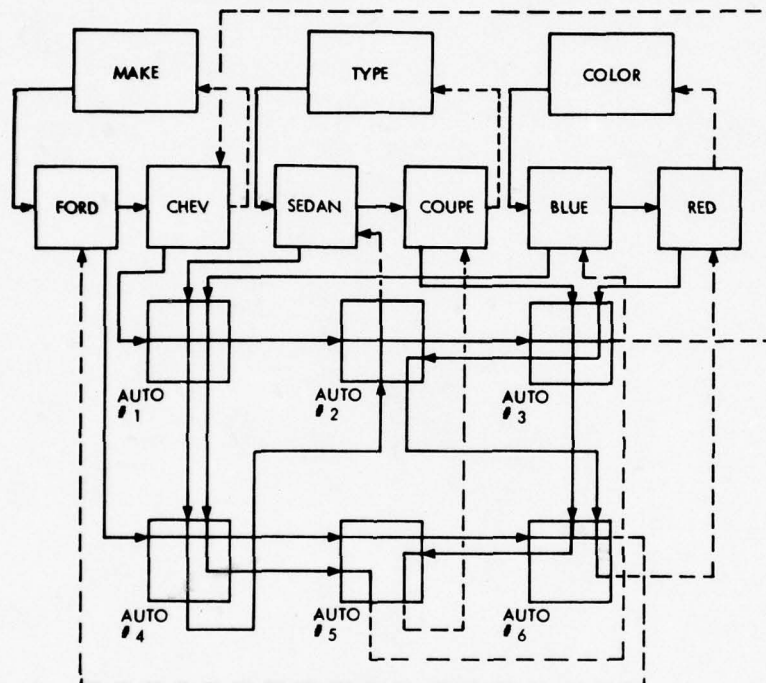
For example, Figure 7-4 describes an I-D-S structure containing the three CALC chains MAKE, TYPE, and COLOR with FORD, CHEV, etc.; SEDAN, COUPE, etc.; and BLUE, RED, etc. as details of the respective chains. In addition, each of these detail records is itself the head of a chain with AUTO records as details. This means that each AUTO record will be linked into chains by manufacturer, model, and color.

The Bachman diagram in Figure 7-4 (a) also indicates that each AUTO detail record carries with it direct pointers to the heads of the respective MAKE-AUTO, TYPE-AUTO, and COLOR-AUTO chains.

Figure 7-4 (b) illustrates the chains and the detail records of each chain. To list all blue Ford coupes, for instance, one need only to follow the COLOR-AUTO chain for BLUE. Then for each element of the chain, check via the direct linkage to the MAKE-AUTO and TYPE-AUTO chains to see whether the BLUE-AUTO is linked into the FORD-AUTO and COUPE-AUTO chain. If the answers are both "yes", list the record; otherwise, step to the next Blue-Auto, if any.



(A) BACHMAN DIAGRAM SHOWING AUTO AS DETAILS TO MASTERS MAKE, TYPE AND COLOR WITH LINK TO MASTER AND CALC MASTER (SEE PARAGRAPH 6.2.1.2).



NOTES:

- (1) DOTTED LINES DENOTE RETURN PATH TO A MASTER CONTAINING, PERHAPS, ADDITIONAL DETAILS.
- (2) IN ADDITION TO THE PATHS EXPLICITLY SHOWN, EACH AUTO DETAIL CONTAINS LINKS TO ITS RESPECTIVE MASTERS.
- (3) MAKE, TYPE, AND COLOR ARE CALC MASTERS.
- (4) AUTO #1 CONTAINS INFORMATION FOR A BLUE CHEVY SEDAN.
 AUTO #2 CONTAINS INFORMATION FOR A RED CHEVY SEDAN.
 AUTO #3 CONTAINS INFORMATION FOR A RED CHEVY COUPE.
 AUTO #4 CONTAINS INFORMATION FOR A BLUE CHEVY COUPE.
 AUTO #5 CONTAINS INFORMATION FOR A BLUE FORD COUPE.
 AUTO #6 CONTAINS INFORMATION FOR A RED FORD COUPE.
 ETC.
- (5) DETAIL DRAWING ILLUSTRATING BUCHMAN DIAGRAM IN (A).

Figure 7-4. I-D-S Structure Containing CALC Chains
MAKE, TYPE, and COLOR

PART III

In Part III, the discussion turns to more recent data base organizational structures: Relational Structures and Normalization Concepts.

Relational structures are logical structures. As such, they can serve as either a model or submodel. Consequently, it is assumed that user submodels are synonymous with the model.

Relational data may be structured in many different ways using the techniques and mechanisms described in Parts I and II, and, since relational structures are models or submodels, no further discussion will be made relating to their storage.

The presentation of relational and normalization concepts, Chapters 8 and 9, has been greatly influenced by the work of Date [F] and Codd [A].

CHAPTER 8 - INTRODUCTION TO RELATIONAL STRUCTURES

8.1 GENERAL

Two mutually incompatible trends in data base development are in evidence. On the one hand, system designers are exposing users to more complicated data structures. On the other hand, they are creating integrated data bases with an increasingly greater degree of interrelated data for on-line interactive usage by non-programming users. According to Codd [J], the introduction of relational data structures, together with powerful manipulation languages, are expected to:

1. Provide a high degree of data independence
2. Provide a global (community) view of the data so that a variety of users within an enterprise can interact with a common data base
3. Simplify the potentially formidable job of data base management
4. Introduce a theoretical foundation into data base management
5. Merge fact retrieval and file management fields to prepare for the addition of inferential services
6. Lift data base application programming to a level in which sets are treated as operands instead of being processed element by element.

The preceding points indicate that the relational approach is aimed at benefiting the end user and the data administrator rather than the application programmer who serves as middleman in today's data processing.

Figure 8-1 illustrates the current trend in system architecture. A relational description of a data base is aimed at providing a description capable of meeting the information requirements for casual and other users. Since it is considered to be a global view of data and capable of expressing all relationships normally encountered in current hierarchical and network structures, it is capable of meeting the information requirements of most users.

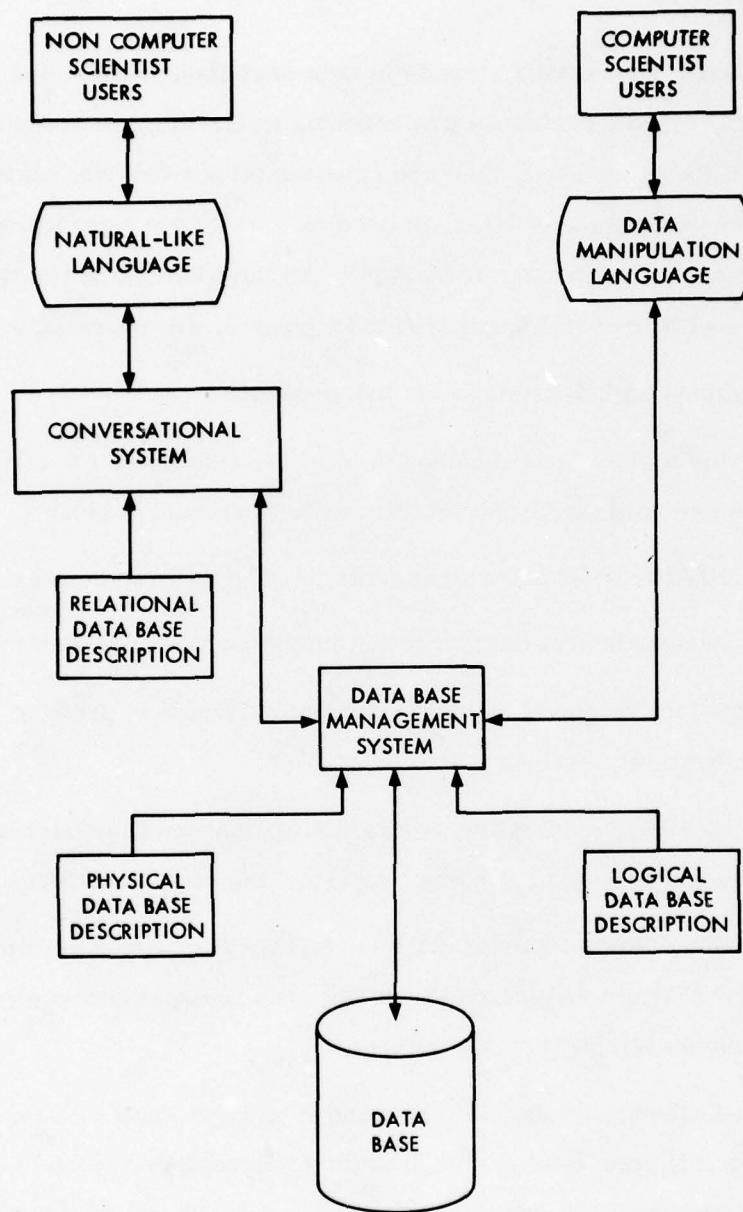


Figure 8-1. Current Trend in System Architecture

The basic ideas and descriptions of essential elements associated with relational models, together with a discussion of relational sublanguages, are set forth in this chapter. Reduction of various data structures in current use to relational forms is covered in Chapter 9.

8.2 ESSENTIAL ELEMENTS

Operational data is a representation of discrete entities of concern to an enterprise. It can be a person, place, or thing. Things may be real or abstract and may include events, classes, and relationships. An entity-set is a collection of entities which possess common properties, e.g., a set of employees. The common properties of entities within an entity-set are referred to as attributes e.g., employee-number, marital status, department number, etc. Associated with each entity of an entity-set is a set of related values for the common attributes. A set of related values is called a record. The set of records associated with an entity-set is a table. The set of all values for a particular attribute associated with an entity-set is a domain. A relation can be thought of as a means for identifying (mapping) corresponding domain elements associated with an entity-set to form records.* Figure 8-2 equates the relational data terminology with the standard data terminology in current use.

8.2.1 Relational Data Base

A relational data base is a data base constructed from a flat arrangement of data elements called data base tables. Each table represents an entity-set.

8.2.1.1 Data Base Table

A data base table is a two-dimensional array with the following properties:

1. Each entry in the table is a simple data item such as a number or character string

*The description for attribute varies from that given by Engles [E], but the description given here is more convenient.

REAL INFORMATION	LOGICAL DATA	
	(AS SEEN BY A USER OF A TREE OR PLEX DATA MODEL.)*	(AS SEEN BY A USER OR A RELATIONAL DATA MODEL).
ALL THINGS IN ENTERPRISE WITH WHICH SYSTEM IS CONCERNED	DATA BASE SYSTEM (CONTAINING MULTIPLE RELATIONAL AND NONRELATIONAL DATA BASES.)	
	DATA BASE (CONTAINING MULTIPLE DATA SETS)	RELATIONAL DATA BASE (CONTAINING MULTIPLE RELATIONAL DATA SETS)
	DATA SET (CONTAINING MULTIPLE DATA FILES)	RELATIONAL DATA SET (CONTAINING MULTIPLE RELATIONAL DATA BASE TABLES)
ENTITY-SET (CONTAINING ONE OR MORE ENTITIES POSSESSING COMMON ATTRIBUTES)	DATA FILE CONTAINING MULTIPLE DATA LOGICAL RECORDS)	RELATIONAL DATA BASE TABLE CONTAINING MULTIPLE RELATIONAL RECORDS OR TABLES
ENTITY AN OBJECT OR RELATION ABOUT WHICH INFORMATION IS STORED	DATA RECORD CONTAINING MULTIPLE DATA AGGREGATES AND/ OR DATA-ITEMS	RELATIONAL DATA RECORD CONTAINING A TUPLE OR A SET OF RELATED DOMAIN VALUES
	DATA-ITEM	DOMAIN NAME
ATTRIBUTE THAT INFORMATION STORED RELATIVE TO A PARTICULAR PROPERTY OF AN ENTITY	DATA-ITEM	RELATIONAL DATA BASE TABLE ENTRY
ATTRIBUTE-VALUE	DATA-ITEM-VALUE	

*SEE PART II

Figure 8-2. Relational Versus Standard Data Terminology

2. Each row of the table is distinct and can be uniquely identified by value
3. Each column of the table contains a homogeneous set of values
4. Each column of the table is assigned a distinct name.

As seen in item 2, the ordering of rows within a table is immaterial since a row can be uniquely identified by value. Item 4 indicates that the ordering of columns within a table is immaterial since a column can be uniquely identified by assigned name. Again referring to item 2, a data base table can be considered a relation between columns of the table, where each column represents a common attribute associated with an entity-set, and each row a set of related values with respect to the common attributes. It is obvious that a data base table is, in fact, an expression of an entity-set.

8.2.1.2 Relation

Generally, a relation is the means for establishing sets of related values, hence, a realization of a relation is a data base table and can be denoted by the n-tuple

$$R (D_1, D_2, \dots, D_n)$$

where R is the name of the relation and D_i where $i = 1, 2, \dots, n$, are column names identifying the respective common attributes possessed by each element in the entity-set. If there are n common attributes about which data is to be recorded (n columns in the realized data base table), the relation is said to be of degree n. A relation of degree one is said to be unary, of degree two to be binary, etc.

A relation which satisfies the properties of Paragraph 8.2.1.1 is said to be normalized. Unnormalized relations are discussed in Chapter 9.

8.2.1.3 Domain

A domain is the set of all values for a particular attribute associated with a given entity-set. Therefore, the set of all values in a particular column of a data base table less duplicates, constitutes a domain. Data base column names are referred to as domain names. Using this nomenclature, the n-tuple

$$R(D_1, D_2, \dots, D_n)$$

denotes relation R on domains D_1, D_2, \dots , and D_n . A realization of R might be the rectangular array:

$$\begin{array}{cccc} D_{11} & D_{12} & \dots & D_{1n} \\ D_{21} & D_{22} & \dots & D_{2n} \\ \dots & \dots & \dots & \dots \\ D_{m1} & D_{m2} & \dots & D_{mn} \end{array}$$

with domain D_k containing elements d_{ik} where $i = 1, \dots, m$ and $1 \leq k \leq n$, less duplicates. The following notation will be used hereafter to define a relation R and describe a data base realization of the relation:

$$\begin{array}{cccc} R(D_1, & D_2, & \dots & D_n) \\ D_{11} & D_{12} & \dots & D_{1n} \\ D_{21} & D_{22} & \dots & D_{2n} \\ \dots & \dots & \dots & \dots \\ D_{m1} & D_{m2} & \dots & D_{mn} \end{array}$$

The same attribute may be used in more than one way within a particular relation; therefore, one or more underlying domains may be actually the same set and the corresponding columns have the same domain name associated with it. To differentiate between these different uses of a domain within the same relation, it is customary to prefix the common domain name with a role name. For example:

$$R(\text{MAJ_}D_1, \text{MIN_}D_1, D_3, \dots, D_n)$$

would define relation R for two uses of common domain D_1 and n-2 other domains. The role name is separated from the domain name by the underscored character.

8.2.1.4 Record

A record is a set of related values which comprise the common attribute values relative to a particular entity included in the entity-set and are indicated by n-tuple $R(D_1 = V_1, D_2 = V_2, \dots, D_n = V_n)$, where V_i are a set of related attribute values. Therefore, a record is a row in a data base table. For example, if relation R is as previously described, $R(D_1 = d_{k1}, D_2 = d_{k2}, \dots, D_n = d_{kn})$, where $1 \leq k \leq m$ describes the kth record of the data base table and $R(D_1 = d_{11}, D_2 = d_{12}, \dots, D_n = d_{1n})$ denotes the first record of the relation. (See Paragraph 8.2.1.3).

8.2.1.5 Key

A key is described as one or more attributes associated with a relation which can be used to identify a record of a relation. A key must possess two properties:

1. Unique identification - value of key must uniquely identify a record
2. Nonredundancy - no attribute in key can be discarded without destroying its unique identification property.

There may be more than one set of attributes which possess the stated properties. Such sets are referred to as candidate keys. The set actually selected is known as the primary key. Practical considerations dictate that a primary key should span the least number of attributes. In theory, a key could, however, cover all common attributes.

In the following examples, key elements will be identified via underscoring. For example, if domains D_1 and D_2 are associated with the respective key attributes for relation R, then

$R(\underline{D}_1,$	$\underline{D}_2,$	\dots	$D_n)$
$d_{\underline{11}}$	$d_{\underline{12}}$	\dots	d_{1n}
\dots	\dots	\dots	\dots
$d_{\underline{m1}}$	$d_{\underline{m2}}$	\dots	d_{mn}

describes the relation and associated data base table. D_1 and D_2 identify the primary key attributes. The primary record keys are d_{k1} and d_{k2} , where $k = 1, \dots, m$.

Throughout the following discussion, all records will be assumed to be keyed records.

8.3 SUBLANGUAGES

8.3.1 Language Characteristics

What kind of operations on relational data structures are of interest to typical relational data base users?*

1. Retrieve a value or set of values
2. Change a value or set of values
3. Insert a record or set of records into a relation
4. Delete a record or set of records from a relation
5. Declare a relation and its domains for inclusion into an established relational data base
6. Add a domain to a relation in an established relational data base
7. Drop a relation from an established relational data base
8. Drop a domain from a relation included in an established relational data base
9. Reduce interaction between himself and an established relational data base
10. Reference a user-declared workspace as a relation
11. Transfer a relation record by record or enmasse to the workspace

Items 1 through 4 provide standard retrieval and update capabilities with respect to relational data. Items 5 through 8 provide the capability to tailor a relational structure to meet the needs of users. Item 9 allows a user to reference relations

*A summary of the language capabilities provided by Codd [B].

through the use of synonyms. Item 10 allows the user to define relational workspaces. Item 11 provides control over the volume of data sent to the workspaces at any one time.

8.3.2 Language Approach

Relational sublanguages can be described in terms of the level of automation which they provide. In ascending order of convenience they can be categorized as record based, algebra based, or calculus based. With the exception of Items 5 through 11 (of Paragraph 8.3.1), no matter which sublanguage type is used, the result of a query or other action is a relation derived in some way from relations currently integrated into the established relational data base.

For example, Figure 8-4 shows the results of the following queries against the relational structure shown in Figure 8-3:

1. Find job codes (JC) for employees in Department 315
2. Find job names (JN) (as opposed to codes) for employees in Department 315
3. Find department numbers (D#) and names for departments at Systems Division Headquarters
4. For each job type, find job code (JC) and all locations (LOC) where that job type is employed.

Usually, the result can be extracted from a single relation as illustrated by query (1), (2), and (3); however, it may require two or more relations as illustrated by (4).

8.3.2.1 Record Based

The record-by-record approach is a low level procedural approach. The user specifies an operation to be performed together with a set of records as operand through a succession of one or more language statements describing each programming step required to produce a desired result. The record set may be empty or may contain one or more records. Generally, low level procedural sublanguages rely heavily upon the capabilities provided by the host language for iterative scanning and searching.

RJ	JC	JN	STA
	11	SA	870
	32	DBA	600
	13	AP	615
	4	SCY	540

RD	D#	DN	LOC
	315	TSD	SDHQ
	320	STS	SDHQ
	314	SED	IDA

RDJ	JC	D#	QC
	11	315	6
	11	320	9
	11	314	4
	32	315	5
	32	320	2
	13	320	3
	4	315	3
	4	314	4

RC ₁	D#
	315

RC ₂	LOC
	SDHQ

JC = JOB CODE
 JN = JOB NAME
 STA = STATUS
 SA = SYSTEM ANALYST
 DBA = DATA BASE ADMINISTRATOR
 AP = APPLICATION
 SCY = SECRETARY
 D# = DEPARTMENT NUMBER
 DN = DEPARTMENT NAME
 LOC = LOCATION

TSD = TELEPROCESSING SYSTEMS
 DEVELOPMENT
 STS = SATELLITE TRANSMISSION
 SYSTEMS
 SED = SYSTEMS ENGINEERING AND
 DEVELOPMENT
 SDHQ = SYSTEMS DIVISION HEAD-
 QUARTERS
 IDA = IDA BUILDING
 QC = QUANTITY COMMITTED

Figure 8-3. Departments and Jobs Data Model in Relational Form with Two "Constant" Relations

1)

JC
11
32
4

4)

JC	LOC
11	SDHQ
11	IDA
32	SDHQ
13	SDHQ
4	SDHQ
4	IDA

2)

JN
SA
DBA
SCY

3)

D#	DN
315	TSD
320	STS

Figure 8-4. Results of Four Queries on Relational Model
(Figure 8-3)

For example, the statement sequence:

	ROPEN	SP (D#, JC) USING KEY
	MOVE	ONE TO INDEX
NEXT	RGET	NEXT RDJ (D# = '315', END = EXIT) INTO W
		USING KEY
	MOVE	JC OF W TO TABLE (INDEX)
	ADD	ONE TO INDEX
	GOTO	NEXT

where ROPEN and RGET are relational sublanguage statements. ROPEN initializes for record-by-record transmission and RGET transmits the next record to a workspace. MOVE, ADD, and GOTO represent typical host language statements.

This statement sequence represents a low level procedural implementation for query (1), Paragraph 8.3.2.

8.3.2.2 Algebra Based

The algebraic approach is a high level procedural approach. The user specifies an algebraic operation together with one or more relations as operands to produce a relation as a result. Algebra-based sublanguages can be made to provide as much selective power as can be made available by a calculus-based sublanguage. The difference is the degree of automation afforded a user and the implementation effort. Algebra-based sublanguages generally require more effort on the part of users to achieve typical results, whereas calculus-based sublanguages generally are more difficult to implement.

There are many possible algebraic operations. For illustration, only operations projection (Π), join ($*$) and division ($/$) will be considered here. The symbols Π , $*$, and $/$, or words PROJECT, JOIN, and DIVIDE, are used as projection, join, and division operators, respectively.

8.3.2.2.1 Projection

The projection operation creates a new relation by "splitting" a relation in the established relational data base. To project a relation over specified domains (columns), the new relation is formed by dropping domains not required and removing redundant records (duplicate rows or tuples) from the remainder.

Figure 8-5(a) illustrates the projection of relation S over domains DN and LOC. The operation is indicated by:

$$R_1 = \Pi_{RD} (DN, LOC)$$

The resulting relation, R_1 , is defined on domains DN and LOC, which are referred to as common domains to relations RD and R_1 . Figure 8-5 (b) illustrates the projection of relation RJ over domain JC. The operation is indicated by:

$$R_2 = \Pi_{RJ}(JC)$$

The resulting relation, R_2 , is defined on the single domain JC.

a) R_1	DN	LOC
	TSD	SDHQ
	STS	SDHQ
	SED	IDA

b) R_2	JC
	11
	32
	13
	4

Figure 8-5. Two Sample Projections Resulting in R_1 and R_2 , Respectively

Projection is also a means for permuting the columns of an existing relation, simply by operating on a relation over a desired ordering of its domains. Figure 8-6 illustrates the projection of RJ over itself, with the resulting relation R_3 where

$$R_3 = \Pi_{RJ} (JN, STA, JC)$$

The relation R_3 is defined on all the domains of RJ.

R_3	JN	STA	JC
	SA	870	11
	DBA	600	32
	AP	615	13
	SCY	540	4

Figure 8-6. Sample Projection Resulting in R_3

8.3.2.2.2 Join

The join operation creates a new relation from two relations in the established relational data base in which the two relations have one or more common domains (See Paragraph 8.3.2.2.1). To join two relations over common domains, the values in each relation for the common domains are inspected. When all corresponding values for the common domains are equal, a set of records is formed containing the concatenation of each such record in the first relation with each such record in the second relation. Redundant records are then discarded.

Figure 8-7(a) shows the join of relations RD and RDJ. The join operation is indicated by:

$$R_4 = RD * RDJ$$

The resulting relation, R_4 , is defined on all domains in which RD and RDJ are defined. Figure 8-7(b) illustrates a join of relations RDJ and RC_1 , which does not use all of the attribute information. The join operation is indicated by:

$$R_5 = RDJ * RC_1$$

The operation proceeds exactly as in the previous join except only attribute information relating to $D\# = 315$ enters into the join as there is no match for 320 and 314 in constant relation RC_1 .

(a) R_4

LOC	DN	D#	JC	QC
SDHQ	TSD	315	11	6
SDHQ	TSD	315	32	5
SDHQ	TSD	315	4	3
SDHQ	STS	320	11	9
SDHQ	STS	320	32	2
SDHQ	STS	320	13	3
IDA	SED	314	11	4
IDA	SED	314	4	4

b) R_5

D #	JC	QC
315	11	6
315	32	5
315	4	3

Figure 8-7. Two Sample Joins Resulting in R_4 and R_5 , Respectively

It is interesting to note here that projection is an inverse operation for join. For example, let RC be a join of relations RA and RB. A projection of RC over the domains of RA yields precisely RA as a resulting relation.

8.3.2.2.3 Division

The division operation creates a new relation from two relations in the established relational data base in which the divisor relation has only domains which are in the dividend. To divide a relation by another, the records of the divisor must be examined. All records of the dividend in which a divisor record cannot be imbedded are dropped, and the divisor domains are discarded from the remaining dividend records.

Figure 8-8 illustrates the division of relation RDJ by the projection of R_5 over QC, $\Pi_{R_5}(QC) = RQ$. The resulting relation R_6 is indicated by:

$$R_6 = RDJ/RQ$$

where R_6 is defined on domains JC and D#, those domains of RDJ not in RQ.

R_6	JC	D#
	11	315
	32	315
	13	320
	4	315

Figure 8-8. Sample Division Resulting in R_6

Note that the division operation is equivalent to a join followed by a projection: $RDJ * RQ$ yields the extension of R_6 in RDJ , and $\Pi [RDJ * RQ] (JC, D\#)$ yields R_6 . It may be that all further operations on the domain level must be composed of joins and projections, in which case they are of little significance apart from notational convenience.

8.3.2.2.4 Boolean Selection Expressions

Most algebra-based relational sublanguages provide a capability for record selection through boolean selection expressions. Boolean record selection expressions can be provided through:

WHERE condition

where condition contains the record selection criteria*. Figure 8-9 illustrates the use of a boolean record selector expression to limit output of the first illustration in Paragraph 8.3.2.2.1 to include only records which pertain to Systems Division Headquarters. The qualified projection operation is indicated by:

$$R_7 = \Pi RD (D\#, DN) \text{ WHERE } (LOC = 'SDHQ')$$

Relation R_7 differs from relation R_1 in that relation R_1 includes records pertaining to both SDHQ and the IDA Building, whereas relation R_7 contains only records pertaining to SDHQ.

*Date [F] avoids the WHERE condition by using the discrete record operation verbs DIFFERENCE and UNION.

AD-A041 459

COMPUTER SCIENCES CORP FALLS CHURCH VA

F/G 9/2

A COMPREHENSIVE DATA BASE ACCESS METHODOLOGIES DESIGN GUIDE.(U)

SEP 76 J EMORY, S GREEN, R GRIMES, O HASLOP

DCA100-75-C-0029

UNCLASSIFIED

CSC-R493700019-2-2

CCTC-TM-123-76

NL

3 OF 4

AD
A041459



R_7

D#	DN
315	TSD
320	STS

Figure 8-9. Sample Projection with Record Selection

Boolean selection expressions can contain complex conditional expressions. For example, Figure 8-10 gives the results of the following algebraic operations:

$$R_8 = \Pi_{RJ} (JC, STA) \text{ WHERE } (STA < 700)$$

$$R_9 = R_8 * RDJ \text{ WHERE } (D\# \neq 320)$$

$$R_{10} = R_9 / R_6$$

The first statement projects relation RJ over domains JC and STA, resulting in relation R_8 containing information pertinent to status numbers 600, 615, and 540. The second statement joins relations R_8 and RDJ resulting in relation R_9 on domains STA, JC, D#, and QC containing information on job types 32, 13, and 4, not employed in Department 320. The third statement divides R_9 by R_6 , resulting in relation R_{10} on domains STA and QC comparing the status and commitment numbers for job types 32 and 4 in Department 315.

R_{10}

STA	QC
600	5
540	3

Figure 8-10. Sample Complex Example Resulting in R_8

8.3.2.3 Calculus-Based

The calculus approach is a nonprocedural approach in which the user can specify a relation as a result. The Data Base Management System (DBMS) is left to work out the details. Relational calculus is simply the notation for expressing the definition of a relation which is to be derived from the established relational data base.

Language symbols include the usual arithmetic and logical operators, the quantifier symbols \exists (there exists), and \forall (for all). Braces $\{ \}$ are used to indicate a set expression, and a colon is used for "such that". The expression on the left of the colon indicates what is to be retrieved, and the expression on the right is a qualification (or predicate).

Using the preceding notation to express the desired results, the query examples of Figure 8-4 can also be used to illustrate the calculus-based nonprocedural approach.

The expression:

$$W \{ RDJ(JC): RD(D\#) = '315' \}$$

may be read, "Retrieve the set of values in domain JC of relation RDJ such that the corresponding values of domain D# of relation RD are equal to '315'.

$$W\{RDJ.JC: RD.D\# = '315'\}$$

is an alternate form of the expression. The two are equivalent and the result shown in Figure 8-4(a) is placed in the specified work area.

The expression:

$$W \{ RJ(JN): \exists RDJ(RDJ(JC)=RJ(JC) \wedge RD(D\#) = '315') \}$$

may be read, "Retrieve the set of values in domain JN from the relation RJ such that there exists an RDJ record with the same value in domain JC as that in the RJ record and with a corresponding value in domain D# of relation RD of '315'. The alternate form:

$$W \{ RJ.JN: \exists RDJ (RDJ.JC = RJ.JC \wedge RD.D\# = '315') \}$$

is equivalent. The result given in Figure 8-4(b) would be placed in the specified work area by the DBMS.

The expression:

$$W \{ (RD(D\#), RD(DN)) : RD(LOC) = 'SDHQ' \}$$

may be read, "Retrieve the set of corresponding values for domains D# and DN of relation RD such that there exists in domain LOC of the relation a corresponding value of 'SDHQ'. The results displayed in Figure 8-4(c) would be placed in the designated workspace by the DBMS.

Finally, the expression:

$$W \{ (RDJ.JC, RD.LOC) : RDJ.D\# = RD.D\# \}$$

may be read as, "Retrieve all job code and location pairs such that the job code comes from an RDJ record and the location comes from the RD record with the same D# value as the RDJ record". The results displayed in Figure 8-4(d) would be placed in the designated workspace by the DBMS.

8.4 ACCESS CONSIDERATIONS

In general, users define one or more workspaces to serve as buffers between their programs and relational data bases. Such workspaces can be referenced as a relation in Data Sublanguage (DSL) statements, and can be manipulated in any way the host language may allow. Regardless of the direction of data flow, relations always appear in a workspace as a rectangular array with the first row containing the respective domain names satisfying property (4) of Paragraph 8.2.1.1.

Retrieval and storage operations on relational data are associative in nature, i.e., users identify, retrieve, and store relational data by value, never by address. Sequential has meaning only in connection with record-by-record transmission of relations provided in a particular implementation of a DBMS as a practical concession to the community of users. Its principal purpose is to provide users with some control over storage requirements for relational workspaces.

8.5 ADVANTAGES

The relational approach is based on the mathematical theory of relations. It provides a sound theoretical foundation in which all relational theory results may be applied directly to such problems as design of the DSL, study of operational side effects, etc.

The relational concept is a simple set of ideas in which the user's view is reduced to its natural structure (without considering access details). Thus, full time and effort can be devoted to solving problems related to the question at hand rather than to solving problems introduced by the data model and associated DSL.

The DSL is relatively simple and completely data-independent.

8.6 DISADVANTAGES

Since large scale implementations of relational data base systems are few in number, there is a lack of practical experience as to the effectiveness of the relational approach.

Relational data base terminology is taken directly from theory, so that in some environments the users are faced with having to learn new terms for familiar concepts (e.g., relation for homogeneous file, domain for single-valued field, etc.).

To avoid undesirable side effects associated with storage operations and/or undue complexity in the DBMS, a relational data base must consist of correctly normalized relations.

There appears to be considerable overhead associated with the scanning and cross-referencing operations required to search the data base tables when relations are stored as tables. In addition, relational data bases require considerable storage space due to redundant information stored with the data.

8.7 APPLICATIONS

To date, no large scale relational data base model is available on the scale as for hierarchical data models such as IMS (IBM). Figure 8-11 presents a summary of

a number of relational systems in tabular forms. All of these systems are to some degree experimental, and only IS/1 and MORIS provide any sort of data submodel facility. None of these systems insists on any particular level of normalization in its data model other than first normal form (See Chapter 9).

SYSTEM	DATA MODEL	DATA SUBLANGUAGE
LEAP	binary relations	record-by-record*
TRAMP	binary relations	record-by-record
Relational Data File	binary relations	limited algebraic
STDS	n-ary relations	limited algebraic
Cambridge	binary relations	a) record-by-record b) limited algebraic
MacAIMS	n-ary relations	algebraic
IS/1	n-ary relations	algebraic
RDMS	n-ary relations	algebraic
MORIS	n-ary relations	calculus

Figure 8-11. Relational System Summaries

Many data bases which reflect the daily operations and transactions of commercial and industrial enterprises are concerned with non-network applications; hence they are candidates for relational data modeling.

Casual users at a terminal who require "browsing" become candidates as are applications such as language translations, directories, and thesauri.

Finally, relational structures can be used as an internet standard in translating formatted data for host compilers in a heterogeneous computer network (Navathe [L]).

*In the literature, a record-by-record based sublanguage is usually called "tuple-by-tuple" or "single-tuple".

CHAPTER 9 - NORMALIZATION OF RELATIONS

9.1 GENERAL

Concepts supported by current data base systems include repeating groups, hierarchical and plex structures, and cross-referencing structures. According to Codd [A], these concepts provide no additional logical capability beyond those of the relational model. He contends that normalized data base tables are the simplest and most understandable way to represent data in the numerous commercial, industrial and utility data bases. Applications such as studies of transportation networks, computer design, etc., have a natural network structure, and therefore require special considerations not covered herein. This chapter is primarily concerned with normalization of relational structures and conversion of repeating groups, hierarchical and plex structures, and cross-referencing structures.

9.2 NORMALIZATION

Normalization as described by Codd [A], is a step-by-step reversible process of replacing a given collection of relations with relations which have a progressively simpler and more regular structure. Reversibility guarantees that no information is lost in the process. In general, the simplification process is based on nonstatistical criteria.

Normalization:

1. Makes it possible to tabulate any relation
2. Obtains required retrieval capability through a relatively simple collection of relational operations
3. Avoids undesirable insertion, deletion, and update dependencies
4. Prolongs life span of application programs
5. Makes relational model more comprehensive to users
6. Frees retrieval and query capability from undesirable transient data dependencies.

The first two objectives apply only to the first step (conversion to first normal form). The last four apply to all normalization steps.

9.2.1 First Normal Form (1NF)

In Paragraphs 8.2.1.1 and 8.2.1.2 the relational model was defined in terms of the 1NF. That is, each entry in the relation was restricted to a simple item value, such as a number or character string. This is equivalent to saying that the underlying domains contain only simple nondecomposable data values (no repeating groups). Every relation satisfying this condition is said to be normalized. Consider the relation RDIR in Figure 9-1. This relation of relations represents the entire network schema of a company directory. None of the domains contains simple nondecomposable data values - each is a repeating group. In order for RDIR to be normalized, all domains must be reduced to data items, whereupon the schema will be in 1NF.

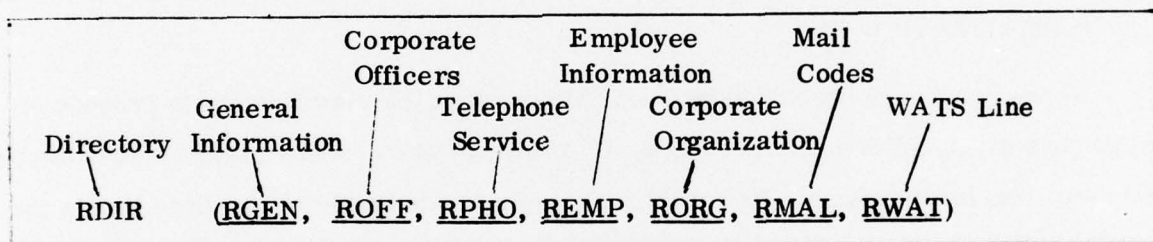


Figure 9-1. Relation of Relations : Unnormalized

To achieve this end, RDIR must be split into its component relations until all are in 1NF. Redundancy is introduced to avoid loss of original information.

Figure 9-2 shows the result of reducing RDIR to its initial domain relations. RGEN and RORG still contain subsidiary relations, but ROFF, RPHO, REMP, RMAL and RWAT have domains consisting only of simple data values and are in 1NF.

RD-GEN (REMERGENCY, RSERVICES)
 RD-OFF (TITLE, DIVISION, EMPLOYEE#, STATUS)
 RD-PHO (TIME, TYPE OF CALL, PROCEDURE)
 RD-EMP (EMPLOYEE#, NAME, EXT, ALTER, BLDG, ROOM, DEPT#, DEPT
 NAME, JOB CODE, JOB NAME)
 RD-ORG (RCORPORATE HQ, RSYSTEMS DIVISION, RSYSTEMS INTERNATIONAL
DIV, RCOMMERCIAL DIV, RCSTA, RENERGY RESEARCH DIV, RFIELD
SERVICES DIV, RINFONET, RPROGRAM DEVELOPMENT, RSYSTEM
SCIENCES DIV, RINTERNATIONAL DIV)
 RD-MAL (OPERATION ID, OPERATION NAME, MAIL CODE, EMPLOYEE#)
 RD-WAT (AREA CODE, WATS LINE NUMBER)

Figure 9-2. RDIR Partially Normalized

The final result appears in Figure 9-3, where each offspring relation has been successively broken down until RDIR is made up entirely of relations in 1NF. Note that each relation is now dependent on its underlined primary keys.

9.2.1.1 Conversion to Relational Structures

The first normalization step is also used to remove repeating groups from non-relational structures and to convert hierarchical and plex structures and cross-referencing structures to relational form. After describing conversion of nonrelational structures to the 1NF, reduction of relations to the second and third normal forms will be discussed.

9.2.1.2 Removal of Repeating Groups

Consider the data concerning job types and the departments which employ them. For each distinct job type, the job code (JC), job name (JN), and status (STA) are recorded together with department data for each department RD using that job type. The department data consists of department number (D#), department name (DN), and quantity (QC) of this job type committed to the department. Figure 9-4 illustrates, in CODASYL fashion, a compound group schema (for jobs) which contains a repeating group schema (for departments).

RD-G-EMER (PROBLEM, PHONE)
 RD-G-SERV (SERVICE ID, SERVICE NAME, ROOM, EXT)
 RD-OFF (TITLE, DIVISION, EMPLOYEE#, STATUS)
 RD-PHO (TIME, TYPE OF CALL, PROCEDURE)
 RD-EMP (EMPLOYEE#, NAME, EXT, ALTER, BLDG, ROOM, DEPT#, DEPT NAME, JOB CODE, JOB NAME)
 RD-O-COHQ (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)*
 RD-O-SYSD (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-SINT (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-CMD (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-CSIA (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-ENRE (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE TELECOPIER, TWX)
 RD-O-FSRV (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-INFT (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-PDEV (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-SSCI (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-IDHQ (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-I-AUST (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-I-CNDA (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-O-I-CSIN (INSTALLATION, ADDRESS, WATS, AREA CODE, PHONE, TELECOPIER, TWX)
 RD-MAL (OPERATION ID, OPERATION NAME, MAIL CODE, EMPLOYEE#)
 RD-WAT (AREA CODE, WATS LINE NUMBER)

*TELECOPIER and/or TWX fields are equal to zero (dummies) when these devices are physically absent from the installation.

Figure 9-3. RDIR in First Normal Form

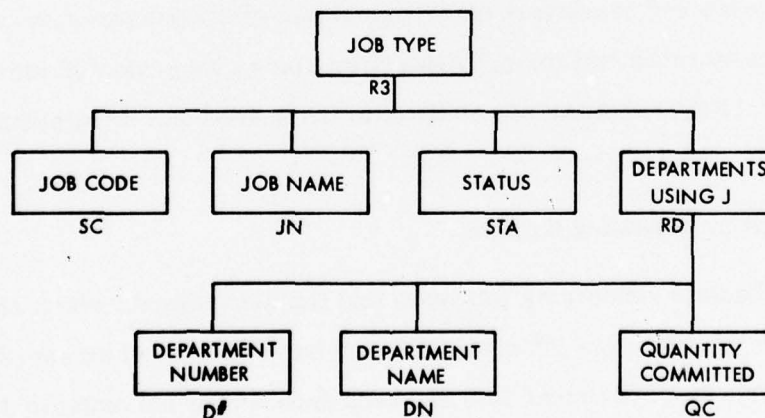


Figure 9-4. Departments as a Repeating Group within Jobs

The simplest way to eliminate the repeating group structure without losing information is to split the schema into two separate schemas as illustrated in Figure 9-5. This split exploits the fact that JC uniquely identifies jobs.

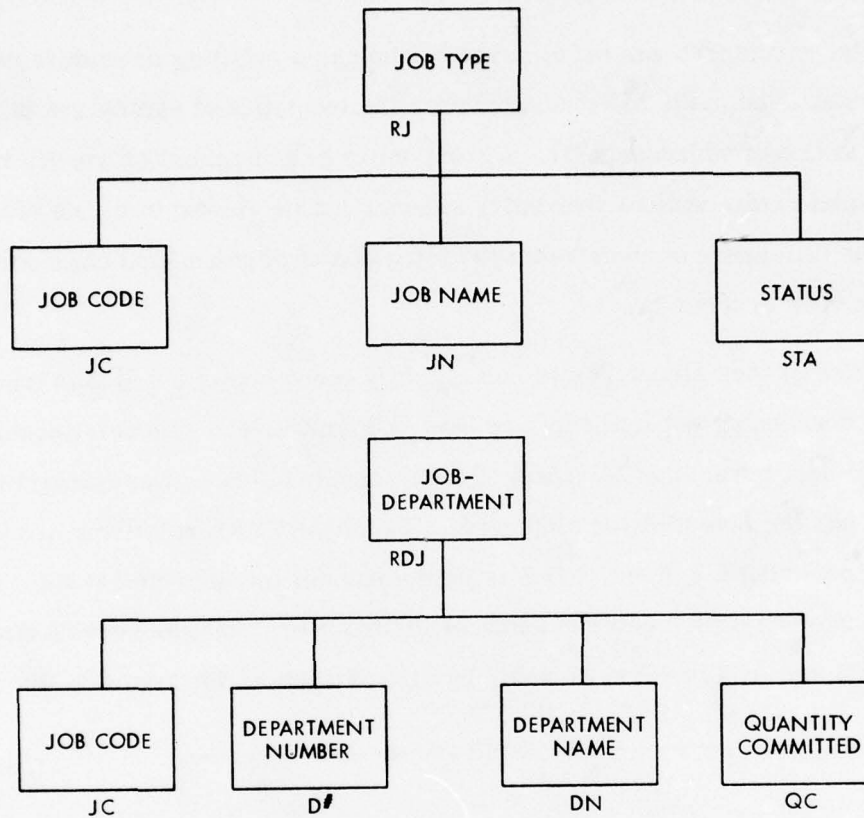


Figure 9-5. Elimination of Repeating Group in Figure 9-4

The schema of Figure 9-5 can be represented in concise relational notation as follows:

RJ(JC, JN, STA)

RDJ (JC, D#, DN, QC)

In more complex situations where two or more repeating groups are subordinate to a common group or in which a repeating group is subordinate to another repeating group, a similar technique is utilized; each repeating group must be split at least once. Note that converting the job relation to 1NF does not resolve all problems. For example,

department information cannot be maintained independently of the job type(s) it uses. A second step in the normalization process (to be discussed in Paragraph 9.2.2) is required to eliminate this short coming.

9.2.1.3 Hierarchic and Plex Structures

Hierarchic structures can be removed by the same splitting procedure used for repeating groups. The main difference between the two types of structures in terms of application is one of addressability. a group entry schema must be viewed by a user as a complete unit, while a tree entry schema can be viewed in terms of its subtrees. This difference demonstrates the intrusion of performance considerations into the user's view of the data.

Elimination of plex structures is only slightly more complicated than trees. Consider the data concerning departments and jobs in Figure 9-6. Two relationships are reflected in the data base: the CANDIDATE relationship holds between department X and job Y if X has the potential for employing a Y; the ACTUAL relationship holds if X is actually employing a Y (i.e., job Y is on the payroll for department X). This data is represented as a plex entry schema in Figure 9-6. Note the representation of the CANDIDATE and ACTUAL relationship by directed lines; this practice is common in

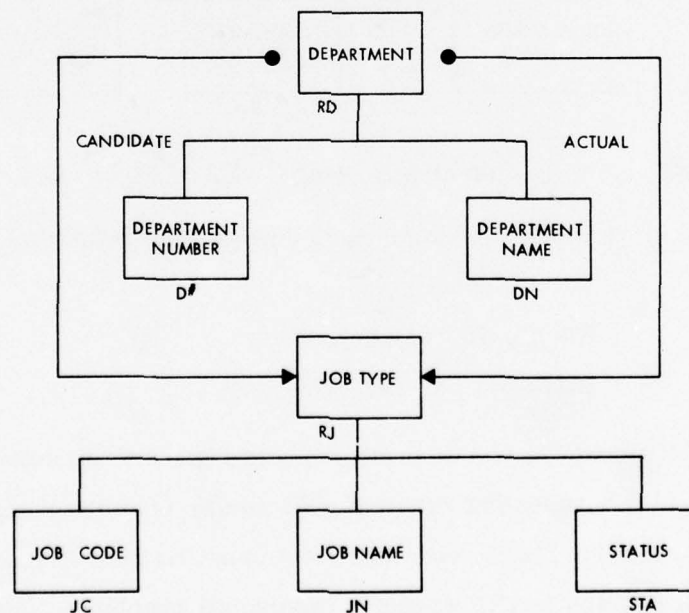


Figure 9-6. Plex Entry Schema

the contemporary literature in this field. It is also another example of physical concepts (pointers) intruding into the user's view of the data.

Each directional relationship* in the plex structure of Figure 9-6 can be split off and represented as a binary relation in tabular fashion. Taking advantage of the fact that D# uniquely identifies departments and JC uniquely identifies jobs, the four separate schemas of Figure 9-8 are obtained.

9.2.1.4 Cross-Referencing Structure

Consider the directory data in relations RD-OFF, RD-EMP, and RD-MAL shown in Figure 9-9. Each corporate officer is an employee and has an employee number. Similarly, the operation manager in RD-MAL has an employee number. In every case, it may be desirable for the user to access a path to RD-EMP in order to derive employee information about an officer or manager.

This referencing connection is shown by one-directional dotted lines, indicating a physical search path to the user. However, this implies that a return path does not exist unless a second link is shown, which results in asymmetry in access. Also, since each link creates a new relation of degree 2, these relations must be treated differently from those of higher degree. They will require different operations for access, modification, security, etc. Further, a linked relation of degree 2 cannot easily be extended to a higher degree.

If the user can have only reciprocal sets of cross-referencing links, his view of the model with all of the possible paths he might wish would become impossibly tangled and complex. For instance, if he wanted employee information on officers and managers, and also a path to find out if an officer were an operation manager, the user would need to be aware of six explicit paths of Figure 9-7 to handle employee number, officer, and manager.

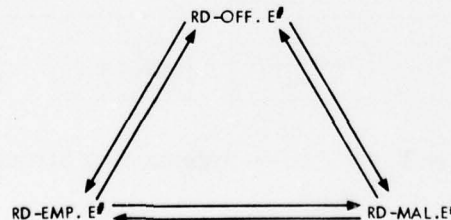


Figure 9-7. Proliferation of Cross-Referencing Links

* The term group relation is sometimes used.

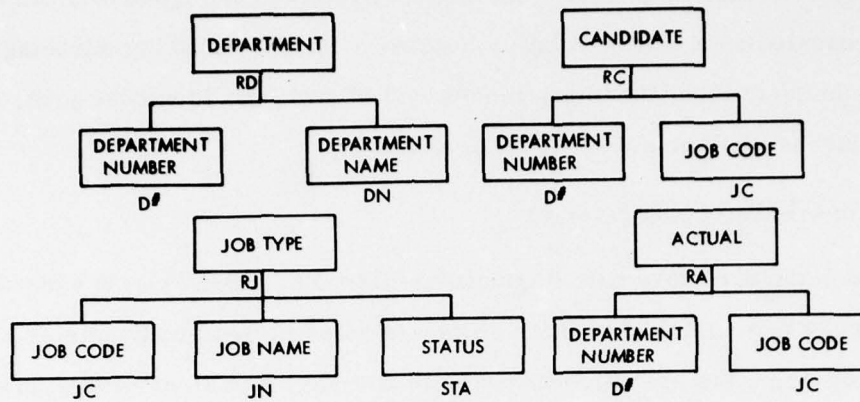


Figure 9-8. Elimination of Plex Structure in Figure 9-6

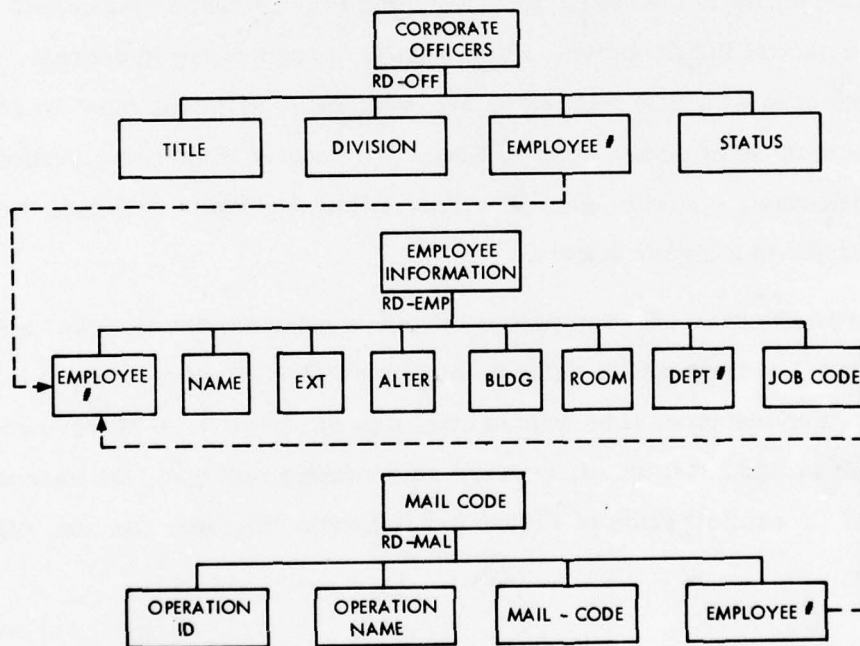


Figure 9-9. Cross-Referencing Structure

This is undesirable and unnecessary, since restrictions as to which pairs of attributes are valid for cross-referencing purposes can be handled by the appropriate language. The concept of system-convertible units fills this need admirably, and entails much the same procedure used to remove repeating groups.

9.2.2 Second Normal Form

As observed in the relation RDJ (Figure 9-5), there are some irregularities in the dependence of attributes upon the primary key (JC, D#). Informally, it is also observed that QC is an attribute of the entire key, while DN is an attribute of the D# component of the primary key. This inconsistency gives rise to the following anomalies:

1. Data concerning a new department cannot be recorded until the department uses some jobs (an insertion anomaly.)
2. If only one kind of job remains in use by a department, deletion of data concerning that job causes deletion of the remaining information on that department, whereas previous deletions had no such consequence (a deletion anomaly).
3. If an attribute value of a department is changed (e.g., the department number), the number of copies of this information to be updated in the data model depends on the number of job types employed by that department at the instant the update is performed (an update anomaly).

The type of irregularity illustrated in this example can be eliminated by splitting the job-department relation, RDJ, to obtain the relations exhibited in Figure 9-10. Note that the QC of a job committed to a department is an attribute of the combination of job code JC and department D#, so it belongs in the relation RDJ_2 , not RD_2 . As usual, the primary key of each relation is underlined. The objective here is to ensure that all nonkey domains are dependent upon the entire key.

9.2.3 Third Normal Form

The final step of normalization is performed to further clarify the PD-EMP relation of the directory. For convenience, an abridged version REMP is shown in

RJ (<u>JC</u> , JN, STA)		
11	SA	870
13	AP	615

RDJ ₂ (<u>JC</u> , <u>D#</u> , QC)	RD ₂ (<u>D#</u> , DN)
11 315 6	315 TSD
11 320 9	320 STS
11 314 4	314 SED
13 320 3	

Figure 9-10. Relations in Second Normal Form

REMP	<u>E#</u>	NAME	EXT	D#	DN	JC	JN)
	1	X	111	315	TSD	4	SCY
	2	Y	121	315	TSD	11	SA
	3	Z	321	314	SED	11	SA
	4	W	324	320	STS	11	SA
	5	V	354	320	STS	13	AP
	6	U	356	320	STS	13	AP

Figure 9-11. A Relation not in Third Normal Form

Figure 9-11. Although REMP has no repeating groups, some attributes are transitively dependent on others. This results in similar operational anomalies to those posed by a relation not in 2NF. For example, DN is dependent on D#, which is in turn dependent on E#. If a department is renamed, this update information must be recorded once for every employee in that department.

Again, splitting the relation removes the problem, and REMP is reduced to REMP, RDEP, and RJOB in Figure 9-12. Now, renaming a department properly results in only one update.

REMP, (<u>E#</u> , NAME, EXT, D#, JC)					
1	X	111	315	4	
2	Y	121	315	11	
3	Z	321	314	11	
4	W	324	320	11	
5	V	354	320	13	
6	U	356	320	13	

RDEP (<u>D#</u> , DN)		RJOB (<u>JC</u> , JN)	
315	TSD	4	SCY
320	STS	11	SA
314	SED	13	AP

Figure 9-12. Relations in Third Normal Form

Finally, Figure 9-13 shows the full schema of the directory in third normal form. All attributes are simply and directly dependent on the underlined primary keys, which yields a structure far cleaner, modifiable, and more readily interrogated than a conventional schema.*

*Codd [A] supplies a vivid contrast of conventional and relational formats of a common schema. (Note, however, that in his conventional structure the cross-referencing links between E# and D# were erroneously omitted.)

Such data models as directories and thesauri do not readily fit into a conventional structure. These schemas contain too much unrelated data for any but a relational format to accommodate efficiently.

RD-G-EMER (PROBLEM, PHONE)
RD-G-SERV (SERVICE ID, SERVICE NAME, ROOM, EXT)
RD-OFF (TITLE, DIVISION, EMPLOYEE#, STATUS)
RD-PHO (TIME, TYPE OF CALL, PROCEDURE)
RD-EMP1 (EMPLOYEE#, NAME, EXT, ALTER, BLDG, ROOM, DEPT#, JOB CODE)
RD-DEP (DEPT#, DEPT NAME)
RD-JOB (JOB CODE, JOB NAME)
RD-O-COHQ (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-SYSD (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-SINT (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-COMD (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-CSTA (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-ENRE (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-FSRV (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-INFT (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-PDEV (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-SSCI (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-I-IDHQ (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-I-AUST (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-I-CNDA (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-O-I-CSIN (INSTALLATION, ADDRESS, PHONE, TELECOPIER, TWX)
RD-MAL (OPERATION ID, OPERATION NAME, MAIL CODE, EMPLOYEE#)
RD-WAT (AREA CODE, WATS LINE NUMBER)
RD-O-ACD (ADDRESS, AREA CODE)

Figure 9-13. RDIR in Third Normal Form

PART IV

Part IV discusses the vital issues of integrity, in Chapter 10, and security, in Chapter 11. Chapter 12, the final chapter, deals with networks and distributed data. A discussion on the composition and organization of networks is followed by a discussion on the storage and access of data which is distributed throughout a network.

CHAPTER 10 - INTEGRITY

10.1 GENERAL

The integrity of a data base management system provides the assurance that the data in a data base is accurate. The system must be capable of safeguarding information within its files. Integrity is usually provided by routines in the DBMS itself in conjunction with the operating system of the hardware being utilized.

10.1.1 Reliability

The major aspect of integrity is reliability. This refers to this system's ability to handle malfunctions of the hardware and software. Hardware malfunctions are usually handled by the operating system, but the DBMS must be capable of recovering from such malfunctions. A software malfunction may occur in the operating system or the DBMS. In either case, the DBMS must be capable of restoring the data base to its condition prior to the malfunction.

10.1.2 Validity

Integrity also refers to the validity of the data in a data base. Validation provides protection of the data from human error. The DBMS can not perform a fool-proof test that a data value is correct, but it can check that the value lies within a specified range or that it is one of a set of permitted values. The DBMS can also check that a value is of the appropriate format.

The validation process of a DBMS may require that certain elements satisfy a given relationship. For example, in the tuple (A, B), the relation $A < B$ may be required.

10.2 LOSS OF INTEGRITY

Loss of integrity may be the result of hardware or software failures. A failure will occasionally occur in any system, regardless of how thoroughly the system has been tested.

10.2.1 Hardware Failure

A modern computer system consists of a great number of hardware components and a malfunction can occur in any of them. In a complex system containing remote computers, a malfunction can occur within the computer at the central site, within a remote computer or in the transmission equipment between the two sites. It is possible for any of these failures to adversely affect a data base stored in the system.

Such conditions as power outage, air conditioning failure, fire or flood can result in a system failure or even total loss of the system.

10.2.2 Software Failure

A software malfunction in any portion of a system can degrade the integrity of the system. The malfunction may occur in the operating system, the DBMS, or an application program accessing the data base. The failure may be the result of a programming error or of incorrect input submitted by a computer operator or terminal user. Any software malfunction presents a potential hazard to a data base. The DBMS must be capable of recognizing when damage has actually occurred. Damage to a data base includes damage to the data or to any directory associated with the data.

10.3 MAINTAINING INTEGRITY

Maintaining a high degree of integrity is an important feature of a DBMS. Data in a data base must be current and correct if it is to be useful. To assist in the maintenance of integrity of a data base, several techniques can be used.

10.3.1 Dump

A data base may be periodically dumped onto magnetic tape to provide a backup copy of the data base. The backup copy should be stored in a safe place. The dumping procedure can be applied to a data base residing on disc or magnetic tape, but most present day data bases reside on disc. The interval between dumps should be determined by the amount of activity associated with the files in the data base.

If a data base is large, dumping of the entire data base is a time consuming operation. An alternative procedure is to dump portions of the data base at shorter intervals, i.e., at a point (checkpoint) at which sufficient data can be stored to permit restarting from that point. If a weekly dump of the data base is sufficient, a portion of it could be dumped each day.

A more efficient procedure is to dump periodically only those files that have been updated. Many files in a data base may be inactive and need not be dumped at frequent intervals. Only rarely should the entire data base be dumped.

With Honeywell's GCOS, an operator may use the PSAVE function to copy designated files, or catalogs of files, from disc to magnetic tape to generate a backup copy of the files. The Integrated Data Store (I-D-S) system also provides a utility routine (QUTD) to dump an I-D-S file, or selected portions of it, onto magnetic tape. A similar utility program (XUTIL) is provided with the Indexed Sequential Processor (ISP).

10.3.2 Journal

A journal is usually recorded on magnetic tape. Information concerning a system's activities is written on the journal while the system is in operation. Journalizing routines are extremely important in a DBMS.

Two types of journal should be kept, a transaction journal and a record journal. However, a system may combine the information required for each and store it in one system journal.

10.3.2.1 Transaction Journal

A transaction journal contains a record of every transaction entering the system. The information recorded for each transaction should include the time that the input was entered, the user's identification number, and a transaction number usually assigned by the system. The transaction journal should contain the full text of the input as entered by the user and may also contain a condensed summary of the requested transaction.

10.3.2.2 Record Journal

A record journal contains actual copies of records in a data base. When a record is to be updated, it is recorded on the record journal both before and after the update occurs. It is also possible for a DBMS to utilize only a "before" journal or only an "after" journal.

I-D-S maintains a journal file on the system's Statistical Collection File or on a magnetic tape, if specified by the user. In addition to transaction information, the I-D-S journal contains "before" and "after" images of each page that is modified. ISP performs a journalizing function when the user supplies a magnetic tape for that purpose. Each time a record is added, deleted, or modified, the page containing that record is written onto the journal as it appeared both before and after the update occurred.

10.3.2.3 Audit Trail

An audit trail can be established which traces items of data through a sequence of processing steps to permit verification of accuracy. In case of failure, the audit trail can aid in retracing the processing steps to a point where recovery can begin.

10.3.3 Recovery

Effective recovery procedures are required to maintain the integrity of a data base. Following a system failure, the DBMS relies upon a successful re-start of the operating system. The DBMS must then determine if the data base has been damaged as a result of the failure.

When a data base, or portion of it, must be restored, the DBMS can use a previously written dump or journal, or a combination of both. If a dump is used for recovery, the data base is restored to its condition at the time the dump was taken. Any updates to the data base since that time are lost. The dump used for recovery should be the most recent one taken. If the most recent one is damaged, the previous one can be used. Two or three backup tapes should be kept; older tapes can be re-used.

For a higher degree of integrity in the data base, a transaction journal can be used in conjunction with the dump. The transactions recorded on the journal are applied to the backup copy of the data base and the most recent version of the data base is produced. This version is the condition of the data base just before the system failure occurred.

In some cases, a record journal can be used for recovery. If the damaged record in the data base exists in the journal, the "after" copy of the record is simply reloaded. This procedure is usually faster than loading the record from a backup tape. A record journal is most useful when a malfunction occurs during the update process. After system re-start, the DBMS restores the "before" copy of the record. If the corresponding transaction appears on the transaction journal, it is applied to the record. Otherwise, the transaction must be re-entered. The current "after" copy of the record is then produced.

Both I-D-S and ISP provide utility routines (QUTL and XJRNAL, respectively) to restore records from a journal. Either "before" or "after" images may be selected. Restoration is performed on the basis of pages, since that is the form in which the journal is maintained.

10.3.4 Detection

Detection routines aid in maintaining the integrity of a data base. The DBMS should use such routines to monitor each transaction before the requested operation is performed on the data base. All add, delete and update operations should be validated. When detection routines are employed by the DBMS, the likelihood of damage to a data base is reduced.

With the add or store operation, a new record is usually added to a data base. The DBMS can verify that the primary key of that record does not already exist in the data base. If it does exist, the new record is rejected. A user may also specify uniqueness for another field of the record, or a combination of fields, in addition to the primary key. If such a condition is specified, the DBMS must monitor all update operations on the data base to ensure that the uniqueness is retained.

In some applications it may be required that particular fields of a record be a subset of a given set of fields in another record. When such a condition exists, the DBMS must monitor delete operations on the data base. The record containing the given set of fields must not be deleted. For example, record type A might contain the courses taught at a particular school, the times the courses are given, and the instructors involved. Record type B might contain the courses taken by a particular student, in addition to other student information. In this case, record type A must not be deleted from the data base.

Detection routines may be used to perform the validity checks discussed in Paragraph 10.1.2. The routines required will depend upon the functions of the application using the data base.

The use of detection routines increases the overhead of a DBMS. Rather than monitor each operation as it occurs, a special utility program could be run periodically to verify the integrity of the data base. If it is found necessary to restore any data, one of the recovery procedures of Paragraph 10.3.3 must be used.

The selection of continuous monitoring or a separate utility program should depend upon the cost of each method and the importance of a high degree of integrity in the data base. It may be that some combination of the two methods would be the most effective approach.

10.3.5 Data Independence

A DBMS must provide some degree of data independence if integrity of the data base is to be maintained. Data independence is the quality in which an application program is unaffected by a change in the structure of a data base or its access method.

Data independence allows the Data Base Administrator (DBA) to control the data base. The DBA may rearrange the data based on frequency of access, factor common data to reduce redundancy, change the representation of data because of new requirements or insert additional elements into the data base.

To aid the DBA, a DBMS should collect statistics concerning the use of the data base and generate utilization reports indicating such statistics. The reports

may be produced automatically or upon request. Utilization reports are generated automatically by ISP and may be obtained for an I-D-S data base with a utility program (QUTA).

Honeywell's WWDMS (World Wide Data Management System) provides a utility routine for restructuring a data base. The routine permits the addition or deletion of record descriptions and data elements. It also permits the expansion or contraction of data elements.

Although data independence provides many benefits, the main concern in this chapter is its effect on integrity. Data independence is of primary importance with shared data; without shared data, it is of minor value. However, one of the reasons for implementing a DBMS is to allow many users to share a common data base.

10.4 APPLICATION CONSIDERATIONS

The techniques used to maintain the integrity of a data base usually increase the overhead of the DBMS. The benefits of this cost are realized in providing accurate and reliable data for those applications using the data base.

10.4.1 Batch Processing

A single-user system is most commonly found in a batch processing environment. In this case, only one user at a time can access the data base. Updates do not occur as rapidly in a single-user system as in a multi-user system, but some integrity features should be incorporated into the DBMS. No system is immune to hardware and software failures. Therefore, backup copies of the data base, and possibly journals, should be kept in the event that recovery of the data base is required. Detection routines should also be utilized to validate all updates to the data base.

10.4.2 On-Line Systems

An on-line DBMS can be either a single-user or multi-user system, although the multi-user approach is more common in present day systems. In a multi-user system, a number of users may concurrently access a data base.

An on-line system must provide a very fast response to a user's transaction. The system must also provide adequate means for maintaining the integrity of the data base. Integrity should not be sacrificed for quick response. If response time is a critical factor, the DBMS could perform a minimum amount of data validation on line and perform more extensive validation of the data base at a later time. Means of rapidly restoring the data base should also be available in an on-line system.

10.4.3 Shared Data Base

A shared data base accommodates multiple concurrent users and is usually accessed through an on-line system. Update operations on a shared data base are extremely complicated and the problem of maintaining the integrity of the data base becomes very difficult.

10.4.3.1 Lost Update

One of the problems resulting from concurrent update of a data base is that of lost update, or double update. If user A and user B each get a copy of the same record, A may update his copy and restore it. User B may then update his copy and restore it. Thus, the update made by user A is lost. A possible solution to the lost update problem is to grant a user exclusive control of the record he is to update. After user A retrieves, updates, and restores the record, he releases control of it. User B may then retrieve the updated copy, with exclusive control, perform his update on that copy, and restore the record. Exclusive control of a record assures that only one copy of the record can reside in main memory for updating purposes during the period of exclusive control. However, exclusive control is not limited to the update process. A user may wish to generate a report and will require exclusive control of an entire file to prevent updates to the file while the output is being generated.

10.4.3.2 Deadlock

Exclusive control, or exclusive access, can create a problem known as deadlock. Deadlock is a situation in which one or more operations are blocked because of requirements that can never be satisfied. If user A has exclusive control of record

X and attempts to access record Y, while user B has exclusive control of record Y and attempts to access record X, a deadlock occurs. Neither user can proceed without some intervention by an operator or the operating system. Most likely, one of the processes will be terminated. A solution to the deadlock problem is to require a user to request at one time all of the records for which he desires exclusive control.

An alternative solution to the deadlock problem is to allow a user to specify either exclusive control or protected control of a record or file, along with the operation to be performed. Exclusive control prevents other users from accessing an area. Protected control allows other users to access the specified area for retrieval purposes only. Updates by other users are prohibited.

The Honeywell File Management Supervisor (FMS) uses the options READ WHILE WRITE, CONCURRENT, and MONITOR to control access by multiple users.

10.4.3.3 Incomplete Update

When a program that is updating a file aborts, or a system failure occurs during the update process, the file may be left in a partially updated condition. Therefore, provisions should be made to protect users from obtaining invalid data from the file. Several means of protection are possible and may be implemented in the DBMS or some portion of the operating system.

If incomplete update is detected for a file, the file can simply be locked. Further allocation of the file is prohibited until the necessary corrections are made. The locking feature does not correct the file; this task must be accomplished by the creator of the file or a user authorized to perform the recovery procedure.

Other means of protection against incomplete update cancel the changes made by the aborted job and restore the file to its condition prior to the start of the job. One of two methods can be used.

In one method, a page (or block) to be changed is written on a collection file. The change is then made to the original file. The same procedure continues until all requested changes to the file have been completed or until the job terminates

abnormally. In the case of abnormal termination, all pages on the collection file are written back onto the original file, cancelling the changes that had been made.

The second method of cancelling changes operates by delaying updates to the original file until the job requesting the updates terminates normally. Each page to be changed is written on a collection file and the change is made to the collection file. When all requested changes have been completed, the pages on the collection file are written onto the original file, producing the updated file. If the job terminated abnormally, the copying of the collection file is not performed.

The Honeywell File Management Supervisor (FMS) uses the options LOCK, ROLLBACK and DELAY to indicate the manner in which a file is to be protected against incomplete update.

Protection against incomplete file update may be provided in a single-user or multi-user environment. However, it is most critical when a shared data base is utilized.

CHAPTER 11 - SECURITY

11.1 GENERAL

Whenever computer systems are discussed, the question of deliberate or accidental compromise of the data contained therein arises. In most commercial or governmental computer systems, the files or data bases contain data which is at least of confidential and often sensitive nature and should be protected from unauthorized access. Governmental security instructions and the Privacy Act of 1974, although designed to restrict data access for different purposes, have the same underlying objective, i.e., to prevent the unrestrained dissemination of sensitive information.

The Government has the responsibility to safeguard the nation's security. A large part of the information gathered to safeguard this security is contained in military and other governmental data bases. This information is constantly threatened and subject to attempts to penetrate the security imposed to protect the information. It is, therefore, important to understand what is meant by security of data bases or computer systems in general, the methods available to protect the information, and the motives and methods used by outsiders to penetrate the security.

11.2 DEFINITIONS

11.2.1 Security

In its larger sense, security of the nation implies all the measures necessary to protect the interest and safety of the nation. However, in the context of this manual, security refers to the precautions taken to guard against the compromise or theft of data or information. One of the methods to safeguard the information/data is to control the access of unauthorized individuals to such information/data. Some of the techniques by which the unauthorized access is controlled are discussed in Paragraph 11.6.

11.2.2 Privacy

Public Law 93-579, 93rd Congress, December 31, 1974, commonly called the Privacy Act of 1974, states that "the right to privacy is a personal and fundamental right protected by the Constitution of the United States". The law was passed to protect this privacy against "the harm to individual privacy that can occur from any collection, maintenance, use, or dissemination of personal information". It allows the individual to prevent records assembled by one agency of the Government pertaining to him from being used by or made available to other agencies without his consent.

11.2.3 Integrity

Integrity has been discussed in Chapter 10. It ensures that the data stored is correct, complete, current, and consistent with pre-established standards.

11.3 FUNDAMENTAL CHARACTERISTICS

The definition of security emphasizes the need for a mechanism by which control of information is maintained. Since there may be many types of controls, their fundamental characteristics must be examined. A detailed mathematical model may provide specific rules for access capabilities. To present a list of generalized properties of security mechanisms becomes difficult when considering the needs of security controls. Browne and Steinauer [Q] present four criteria which encompass the majority of characteristics desired in a security mechanism. These four properties are functional, inexpensive, simple, and providing program generality.

11.3.1 Functional

The security mechanism must provide protection of the user's information through access control capabilities. The protection must be capable of controlling access to information without imposing undue restrictions upon the user. Access authority must be checked continuously for all data access. The protection controls must be uniform for all possible functions.

11.3.2 Inexpensive

Security control procedures usually require additional processing steps. The amount of overhead required to implement security checks is proportional to the extent of control desired. Without reducing security effectiveness, emphasis must be placed upon minimizing overhead and therefore cost.

11.3.3 Simple

Whatever the security precautions, the control procedures must be simple. The procedures, to operate correctly, should avoid undue complexity, i.e., the software access programs should be kept simple, yet effective, to prevent unauthorized data access and penetration.

11.3.4 Program Generality

Procedures to provide security control should be transportable between systems. Reusability of security controls allows that reliable procedures and/or programs can be utilized for a variety of applications or computer systems. Not only is this cost-effective, but it provides a high level of confidence that the security controls are reliable.

11.4 MATHEMATICAL MODEL

A detailed mathematical model, as that described in Appendix A, can provide a better understanding of the security mechanisms required. The ability to define security in detail provides the designer with a tool to examine the various control techniques. The processes are straightforward once the basic definitions have been made.

11.5 SECURITY COMPROMISES

The degree of security controls necessary to protect vital data depends upon many factors: size of the system, sensitivity of data, accessibility, etc. The advent of the computer has presented a whole new opportunity and risk for potential compromise situations. Therefore, new techniques must be found to reduce the risk.

11.5.1 Causes

The level of security controls required is a function of the threats which may exist. There must be a method by which the threats can be defined and a value placed upon each. Analysis of the causes and associated costs is a prerequisite for evaluating potential risks. A risk analysis lists the possible threats, and ranks the threats in a manner which allows a cost-benefit analysis. The results of the risk analysis can be used to present potential dangers to management so that decisions can be made as to what level of security control is cost-justified.

11.5.1.1 Risk Analysis

Risk analysis* procedures are simple; however, the complete analysis is more complex. The first step in risk analysis, the information gathering phase, requires a detailed analysis of all possible threats to the system in question. Figure 11-1 indicates typical threats. The next step determines the number of times per year(s) the threat may possibly occur. In addition, a dollar amount must be determined for each threat on a per event basis. The dollar figure associated with each threat will vary considerably with the installation (Note: numbers used in the example are arbitrary). For simplicity, Figure 11-2 shows a few possible threats and their cost per event. From Figure 11-2, the graph in Figure 11-3 (1 of 2) can be derived. The X axis of the graph represents the events per year, scaled logarithmically. Each threat is indicated as an X, Y pair. The final step in the risk analysis is to determine the product of the two quantities (events per year and dollars per event). This can easily be done on the graph by inserting the diagonal lines, as indicated in Figure 11-3 (2 of 2), which represent the products of X and Y axis coordinates. The graph can then be tilted 45 degrees and the ranking of potential threats is indicated by the position of the points relative to the new horizontal lines. Management can easily determine which potential threat will cost the most.

* The graphical risk analysis is used here; however, a numerical analysis which results in the same conclusions can be utilized.

<u>Physical</u>	<u>Software</u>
Fire	Application Failure
Sabotage	Utility Failure
Air Conditioning Failure	DBMS Failure
Flood	Operating System Failure
Electrical Failure	Communication Software Failure
Break In	Software Penetration
Earthquake	
<u>Operational</u>	<u>Hardware</u>
Operator Error	Mainframe Failure
Keypunch Data Error	Tapping Communication Lines
Mischief	Incorrect Hardware Operation
Theft	Substitution of Hardware Devices
Espionage	
Misdelivery of Secured Information	
Improper Data Handling	

Figure 11-1. Typical Threats

<u>THREAT</u>	<u>EVENTS/YEAR</u>	<u>COST/EVENT (\$)</u>
(○) Personnel Negligence	1000	100
(□) Power Failure	3	1000
(Δ) Break In	1	1000
(⊂) Air Conditioning Failure	1 to 3	3K
(☆) Fire	1 to 30	1M

Figure 11-2. Potential Threats and Associated Costs

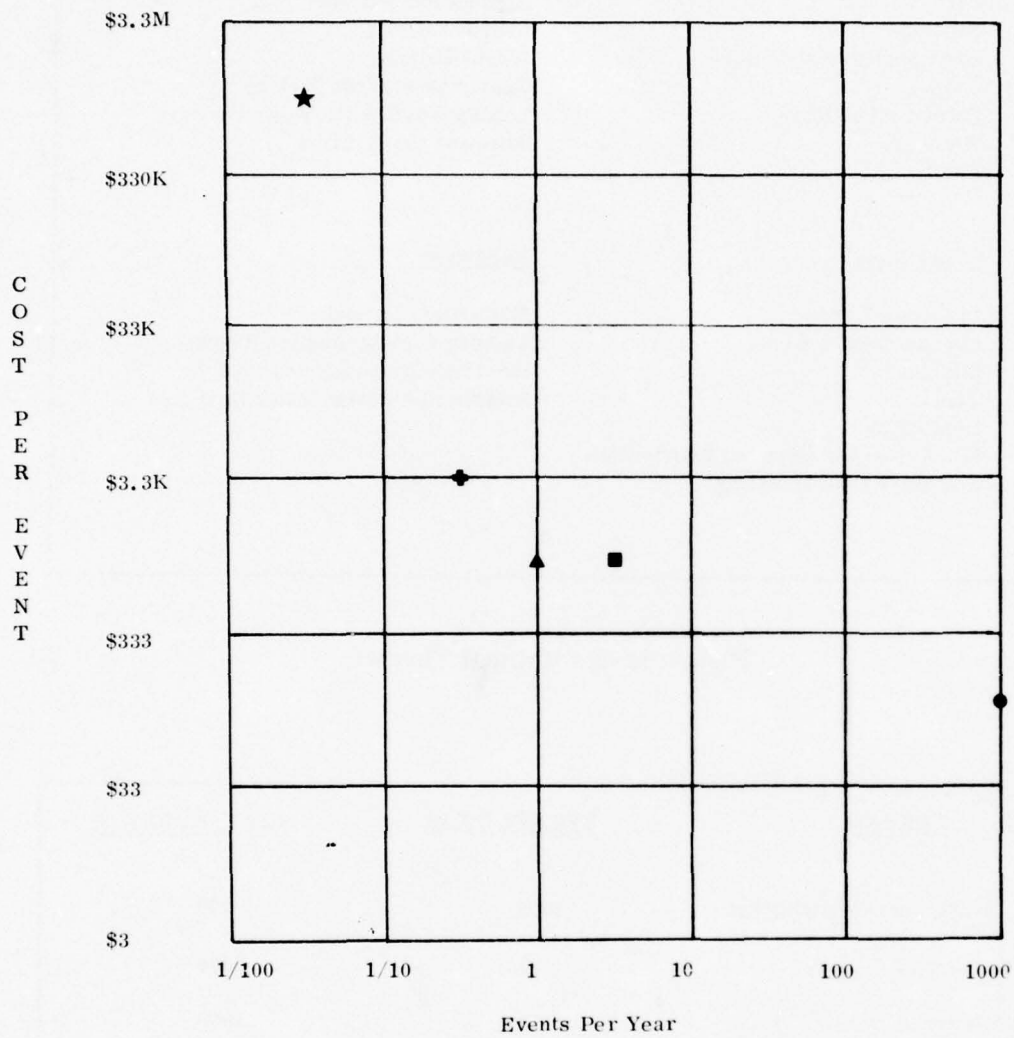


Figure 11-3. Risk Analysis (1 of 2)

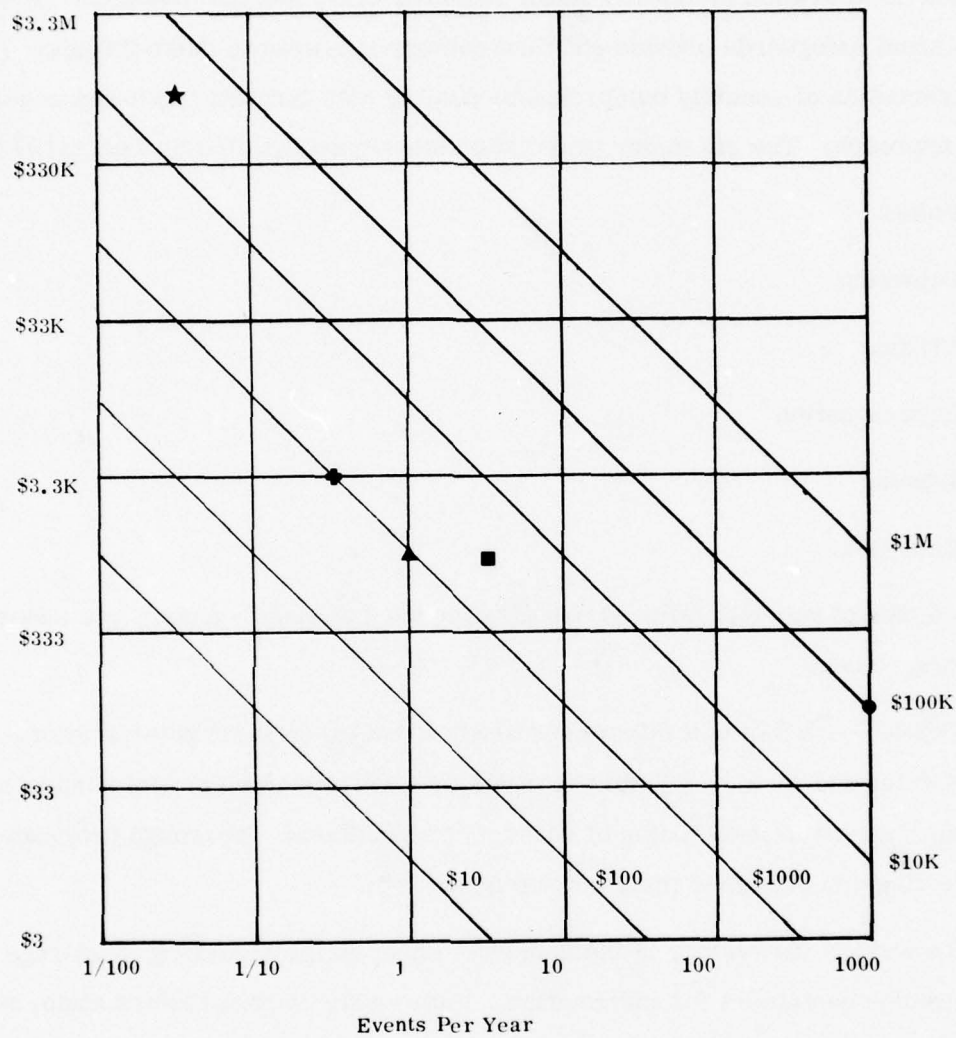


Figure 11-3. Risk Analysis (2 of 2)

11.5.1.2 Penetrations

Physical security breaches such as unauthorized building or vault entry, sabotage, etc., can effectively be controlled through human intervention. Installation of security systems such as television monitors, badge readers, etc., and the insistence on adhering to manual safeguards provide efficient countermeasures to these threats. However, the prevention of security compromises dealing with computer systems requires a different approach. The six major penetration techniques detailed by Lackey[U] are:

1. Foible
2. Browsing
3. Artifice
4. Impersonation
5. Tapping
6. Radiation.

Specific examples of possible failures are also included to ensure a complete understanding of the causes.

1. Foible - A foible is a minor flaw or shortcoming in a computer system. Foibles are introduced into programs by either incomplete design specifications or inadequate coding of correct specifications. Thorough program testing can eliminate their occurrence.
2. Browsing - Browsing, in the security sense, is the searching of storage or residue containers for information. Improperly disposed information, such as passwords, access lists, file names, etc., can provide an easy access to unauthorized information. Tape and disk may contain bits of classified information if not properly erased.

3. **Artifice** - An artifice is the intentional inclusion of code within the program which will allow subversion of the system. Artifices may be included in programs either by unscrupulous programmers/designers or through a penetrator's program modifying other programs. Most artifices operate so that the program functions as specified but also performs an additional unauthorized function. This type of operation is generally known as the "Trojan Horse".
4. **Impersonation** - An impersonator is an individual who gains access to secure information by assuming the characteristics of an authorized user or device. Impersonation is usually combined with other penetration techniques, such as browsing, to provide access to the system. Browsing through waste material provides the userid/passwords necessary to be identified to the system as a legitimate user. The detection of impersonators is difficult, since all validation checks are correctly completed.

Impersonation may also include hardware devices. A penetrator may substitute his own device. Unauthorized terminals may be connected between the main site and the legitimate terminal.
5. **Tapping** - Tapping is the ability to access a system by a direct connection to a communication line or part of the central system.
6. **Radiation** - Radiation is a method by which information is gathered without a physical connection into the system. The penetrator can utilize sophisticated electronic equipment to detect acoustic or electromagnetic radiation from computer systems or devices. Communication lines, cathode ray terminals, and line printers are highly susceptible to eavesdropping by radiation.

11.5.2 Specific Hardware/Software Examples

Penetrations can be categorized as hardware and software. Although general weaknesses can be described, actual hardware/software systems may have their own problem areas.

11.5.2.1 Hardware Error Detection

With hardware, most problems which allow security breaches are not easily detected, especially when the detection mechanism fails. The majority of latter generation architectures can detect their own errors and correct them so that a successful operation will be completed. However, it is necessary to examine the error correction techniques so that a security breach does not occur during this process. Most systems will log all unsuccessful tries; however, successfully corrected errors are usually not recorded. During a successful error correction, the actual security breach may occur.

11.5.2.2 Replacement of Hardware Components

The placement of unauthorized hardware components must be carefully monitored. The operating system cannot detect when an unauthorized device is placed on the system, since most hardware devices do not have unique device identification. Devices may provide the means for a penetrator to gain access to unauthorized information. Therefore, maintenance personnel should be carefully screened.

Hardware devices also can be impersonated. For example, a terminal may have a unique identification such as an answer back drum. The impersonator may find it easy to duplicate the device and the central system will not be cognizant of the impersonator's terminal being connected. Therefore, unalterable or nonduplicatable device identification should be utilized.

11.5.2.3 Lack of Multiple Domain Architecture in Operating Systems

Effective operating systems have certain privileges which the user programs do not have. Such privileges as initiating I/O, memory management, resource allocation, and security enforcement are functions of the operating system performed on

behalf of all users. These functions must be delegated to the operating system for efficient and orderly processing of user programs. To provide this distinction, many hardware architectures allow a bi-state operation. The master, or privileged, state allows the execution of certain instructions which are not allowed in the slave or non-privileged state, thus, allowing access to all memory, peripherals, and security controls. The user program cannot access these functions directly. This protection can be provided by the extension of the bi-state to multiple states called domains.

The type of architecture which utilizes domains provides the capability of user programs executing in higher domains, thus allowing the operating system to operate in lower domains. The key to successful implementation of this type of system is that the entire operating system does not operate in the same domain. For example, in Figure 11-4 the user programs may operate in domain 4, utility functions in domain 3, scheduler in domain 2, peripheral allocator in domain 1, and security in domain 0. Once the domains have been defined and the code written for each, the ability of one domain to obtain information from another can be carefully controlled. The lowest domain has access to higher domains, whereas a higher domain cannot access information in the lower domain without proper identification*. The lowest domain, the kernel, actually does such functions as I/O initiation, memory management, and security checks.

11.5.2.4 Parameter Processing.

Parameter processing during transition between states is a serious problem. Parameters must be validated when calls upon lower domains are requested so that the request is proper. However, within some architectures, the ability to change the parameters after the call is made may permit a breach of operating system security. Access to unauthorized portions of memory or restricted I/O may be accomplished through modification of parameters. The ability to request a function of a lower domain may be legitimate within multiple domains. However, the parameters and data areas

* See Appendix A describing interprocess access.

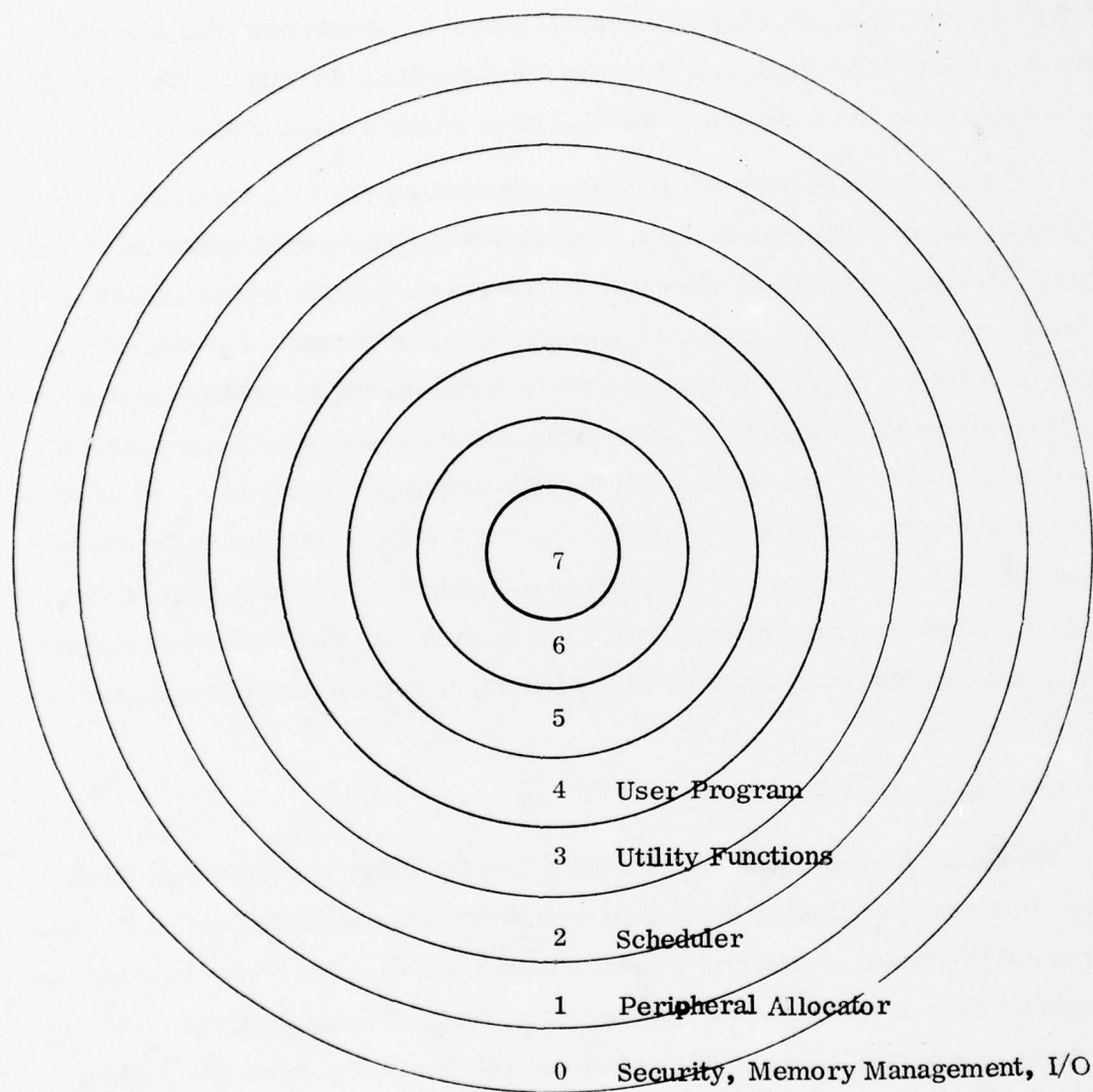


Figure 11-4. Domain Structure

to be accessed must be validated so that the domains may be crossed correctly. The largest problem with multiple domain designs is that the pointers to parameters and data areas can easily be modified, thus causing security breaches. This type of breach is known as the "leaky parameter problem".

11.5.2.5 I/O Control

The multiple domain machines have provisions which restrict physical I/O. Requests for physical I/O may allow access to sensitive information either through incorrect parameter checking or weakness in design. Potential security breaches due to physical I/O are decreased in virtual memory systems. Virtual systems provide the capability to operate programs and store data in a user's workspace. The initiation of I/O is delegated to the kernel, thereby ensuring that other user programs and other parts of the operating system cannot access the restricted stored information.

11.5.2.6 Utility Software

Many operating systems provide utility software to perform functions such as disk dumps, file transfers, file modifications, etc. These programs must be carefully examined so that usage by an unauthorized user cannot compromise any information in the system. Many programs are written by system programmers to allow modification of memory locations or disk control tables during system operation. These programs must be carefully controlled so that confidentiality of the data is maintained. Also, certain types of debug packages provide the ability to examine memory locations, buffers, etc., which may simplify the penetrator's ability to access sensitive information.

11.5.2.7 Lack of Security in Design

The fundamental weakness of all commercial systems is that their basic design was not built upon security requirements. Security controls were an add on and, as such, do not provide the complete controls needed. Such a highly rated system as the Honeywell Multics was designed for access control [S], not security control. However, new architectures are being designed with security in mind. The ADEPT-50

time-sharing operating system [T] was designed with consideration of security requirements. Future systems, such as the Honeywell Level 66, will provide some combination of both hardware and software security requirements.

11.6 METHODS TO MAINTAIN SECURITY

This paragraph discusses the methods by which a user or designer can protect himself against security compromises.

Three methods can be used to obtain a secure system:

- Minimize probability of a security breach
- Minimize damage if a breach occurs
- Institute an adequate recovery plan for a disaster.

Some factors cannot be controlled by a programmer or hardware designer, but the future system architecture should provide this control.

11.6.1 Hardware Checks

Specific hardware design must be included within the system to enforce certain types of security checks. A memory protection mechanism can be designed which consists of a set of hardware registers. These registers will be utilized for each memory access to ensure that the requested data lies completely within the user's data space. Similar hardware registers should be incorporated into the I/O Controller hardware.

Hardware may interpret specific portions of access control words specified by the software. The hardware's ability to examine specific bits or patterns for match on a given key restricts users from requesting memory locations which do not belong to *their data space*. Multics has implemented such a mechanism to assure that the user is a member of the access list for the domain in which he is processing.

A serious weakness with hardware structures is the bi-state processor. Multiple domains with adequate protection between domains and controlled movement between domains will provide proper security. Systems such as Multics on the H6180 processor

provide this capability. The concept of multiple domains was originally implemented on a bi-state processor (GE645); however, the required overhead made the system inefficient for processing.

Future systems will provide redundant hardware to assure that processing functions are working correctly. Redundant processing will provide detection of incorrect operation and immediate reconfiguration eliminating any loss of processing time. Two or more CPUs working in parallel will provide the capability to enforce security controls. It will be difficult for a penetrator to place his own components in systems of this type. Also, the obvious benefit of more reliable systems will dictate such an approach.

11.6.2 Data Locks

The model defined in Appendix A uses access lists to indicate the permissions allowed to specific objects. The inclusion of access lists for objects such as programs and files can provide software control. The lists for permissions are kept separate from the data to prevent their alteration by unauthorized users. There is a distinction between the permissions for data access (i.e., read, write, execute, etc.) and the control permission (i.e., modify). Normally, the control of access lists is the function of the File Management System*. For control over more detailed objects, such as data fields, or items, program logic may be included. Lists of this type are controlled by the programmer.

Access lists may be combined to form authorization tables which group users by some specific criteria. The manipulation of these tables provides the programmer with the ability to determine which user, or group of users, has access to particular data items. By combining similar attributes and utilizing bit maps (Paragraph 6.3.4), efficiently organized tables may be stored.

* Honeywell 6000 File Management Supervisor provides access permissions for each file or catalog within the permanent file structure.

The domains implemented by the hardware function need certain software control. The ability of software to utilize domains provides protection of data elements. Not only are unauthorized users prevented access, but the sharing of data can be efficiently accommodated.

11.6.3 Passwords

Historically, the most common form of security controls have been passwords. A password is an identifier used as part of a validating sequence.

In the past, the need to identify a legitimate user could easily be accomplished by a visual contact. However, with the advent of time sharing and remote processing, personal recognition based on visual procedures is no longer practical. A means for positive identification of an authorized user over long distance communications lines had to be found. The log-on sequence with user identification (USERID) and password was the consequence.

To frustrate a penetrator, the password must be some random pattern of characters. Also, the full length of a password capability should be utilized. Whenever a remote user enters a password from a remote terminal, the password is usually printed or shown on the input device. This printout may allow an unauthorized person to obtain another user's password. To protect the password, system designers have provided a strike-over mask* for the space where the password is to be printed. The best alternative to eliminate a security compromise is to use either nonprinting passwords (control characters not belonging to the alphabet) or to disable the print mechanism on the terminal.

The nondisplayed password does not prevent a penetrator from tapping the communication line and decoding the password. A more complex means of preventing a breach of this type is by using one-time passwords. A list of passwords is generated for each user. The user then uses this list to determine which password to use for the next log-on sequence. Once a password is used, it cannot be used again.

* A series of random characters printed over itself to hide the following typed input.

An alternative for password lists is to use timed passwords. A series of passwords are given to the user with an associated time interval through which the password is valid. The usage of a password outside its time interval will alert the security controller that a possible penetrator is attacking the system. Also, there may be intervals in which the user is restricted from the system allowing a higher classification of work to proceed.

All of these methods of password control use a table within the computer system (usually on a protected system disk). Since tables may be subject to penetration by unauthorized persons, a measure to protect passwords has been developed through the utilization of a one-way transformation [R]. An algorithm can be devised which will produce a unique result based on a given password. However, the application of the algorithm in reverse will not yield the original password. With this method, penetration is prevented; however, if the user forgets his password, it cannot be obtained from the transformed table.

An important deterrent to potential penetration is variation of the password or portions of it. A user who frequently changes his password does not allow the penetrator sufficient time to guess or steal a password. Systems which allow users to change passwords dynamically provide a greater degree of protection. Also, validation processes which require random portions of a password, decrease the probability of a penetration. Additional personal information, other than a random pattern of characters, provides even more protection. Question-answer sequences may be used to interrogate the user to obtain personal information.

Mechanical identification methods such as magnetic badge readers are also used to provide physical security. Other identification devices measure individual characteristics such as fingerprints, voiceprints, and relative finger length. These mechanical devices provide a great degree of protection.

11.6.4 Encryption

Encryption is the application of a set of mathematical rules to render information unintelligible. The rule, or algorithm, transforms the plain text of a message into a series of symbols by the application of a key. The same key must be applied to obtain the plain text. There are many methods of applying encryption to data streams, through both hardware and software modules. The fundamental properties of an encryption mechanism must be an integral part of the design, otherwise the technique will not be cost-effective. The method by which encryption is implemented must be simple. The algorithm should be designed to prevent the penetrator from deciphering the code. Protection through encryption relies on the secrecy of the key and the key should be easily changeable to allow frequent replacement. Finally, the fact that encryption was used should be completely transparent to the user.

The best, and most frequently used, method of encryption is based upon a repetitive addition of the plain text and a specified key. Only by knowing the key, or decryption algorithm, can the scrambled text be translated into plain text. The most secure technique uses a once only key, changed after each encryption. Alternatives to once only keys are keys generated by some random generator, thus giving a list of keys. If the keys are changed frequently, the penetrator will not have enough time to decipher the message. Multiple keys may be used to produce two levels of encryption, thereby confusing the penetrator if he does succeed in obtaining one key.

Most applications place the encryption device as near as possible to the terminal and the decryption mechanism at the computer front end. This type of encryption provides communication security, but does not protect the data once it is deciphered. Since most computer hardware cannot process arithmetically encrypted data, the information must be in plain text during processing. Encryption devices may also be placed between the central processor and the actual storage media. As long as the key is secure, the penetrator cannot gain access to the data.

With the advent of microprocessors, encryption devices may be inexpensively installed at storage or processor interfaces. Future systems may even provide capabilities to process encrypted data through all hardware devices.

11.6.5 Combination of Methods

The key to effective security controls is to lower the probability of a penetrator gaining access to information. By increasing the amount of work required by the penetrator, the likelihood of compromise is decreased. Therefore, the required "work factor" must be determined for each type of access. Increasing the work factor decreases the probability that the penetrator will gain access.

11.6.6 Monitoring and Surveillance

The utilization of monitors and surveillance techniques does not directly contribute to the prevention of security breaches; however, it does provide information for the detection and apprehension of security violators. Monitors and surveillance techniques (henceforth referred to only as monitors) can provide useful data by including them into all phases of security control. The monitor may be a software module, or program, which collects information relative to possible security breaches.

Monitoring, as described by Barteck [N] , can be implemented in two different ways: active or passive.

Active monitoring is the inclusion of modules which constantly look for security compromises. The monitor activates the proper alarm, usually at a security control console, once a breach is detected. The alarm may be in the form of a message type-out, activation of a bell, or activation of a flashing light. Whatever mechanism is used, the alarm must attract the attention of the security officer.

Once the alarm has been activated, automatic action may follow. If the compromise came from a program, the program may be suspended or aborted so that security personnel can investigate why the program acted as it did. For terminal-oriented violations, the monitor may lock out the terminal, either by a software lock-out or by sending a command which disables the terminal. In either case, the concept

is to prevent the user from trying to regain access. Monitors which activate sophisticated alarms and terminal lock-out may provide enough time and information for the security officer to locate the penetrator*.

In passive monitoring, information is gathered during the violation, but no action is taken. The detection of a violation is still required, followed by a recording of such items as changes in users' hardware/software status, security parameters, and access to sensitive information. For all security related events, whether violations or not, information concerning:

- All information access and/or denials
- Log-in/log-out sequences
- Violation of memory bounds and privileged instruction usage
- Change of access control profiles
- Change in system configuration
- Transmission of sensitive data
- Security parameter changes

must be recorded so that later processing can be done.

The final aspect of passive monitoring is the analysis of the data recorded during the system's operation. Reports must be generated which provide information to detect possible penetrations**.

* WWMCCS Honeywell operating system utilizes an alarm (message type out, activation of audio signal, and flashing light) at the operator's console. Programs are aborted and terminals are locked out for detected violations.

** WWMCCS provides reports from accounting and security data (program GESEP) for security personnel to examine.

11.7 OTHER SECURITY CONSIDERATIONS

There are other events which may require security investigation depending upon circumstances. One of these is recovery. A recovery plan should be an integral part of a complete security design. Also, network security must be considered if distributed data bases or processes are to be used.

11.7.1 Recovery Plan

An adequate recovery plan must be designed for a complete loss of information. Recovery plans require periodic copying of sensitive data, to be placed in an off-site storage facility, but this is only part of an adequate plan. Not only data, but programs, documentation, supplies, and transaction histories must be stored safely and securely. Just as a risk analysis is an integral part of security planning, the design of the recovery plan must also be an integral part of security planning.

11.7.2 Distributed Data Bases

The introduction of data bases distributed over many computers connected by a communications network provides many new challenges to security. The typical distributed data base is utilized by many users with varying data processing requirements. The more users, the higher the probability for a penetrator to masquerade as a legitimate user. Complex security controls must be carefully implemented to protect the data. The use of heterogeneous devices within the network complicates the control mechanism.

Even though the complexity of security increases, the probability of a violation is decreased by keeping portions of the data base on different systems and possibly periodically moving them.

11.8 APPLICATIONS

A description of all security controls for all DBMS would be lengthy. Therefore, only the standard DBMS proposed by the Data Base Task Group (DBTG), Integrated Data Store (I-D-S), and World Wide Data Management System (WWDMS) will be presented here. The controls described are only those provided by the language and not those of the operating (or file) system.

11.8.1 DBMS Proposed by DBTG

The DBTG DBMS imposes control through codes assigned to values. The user must give the proper code to allow access to the restricted value. Also, a security validating procedure can be associated with an item reference. The actual syntax for implementing the control is shown in Figure 11-5. The security locks may be imposed at the file, entry, group, and item levels*.

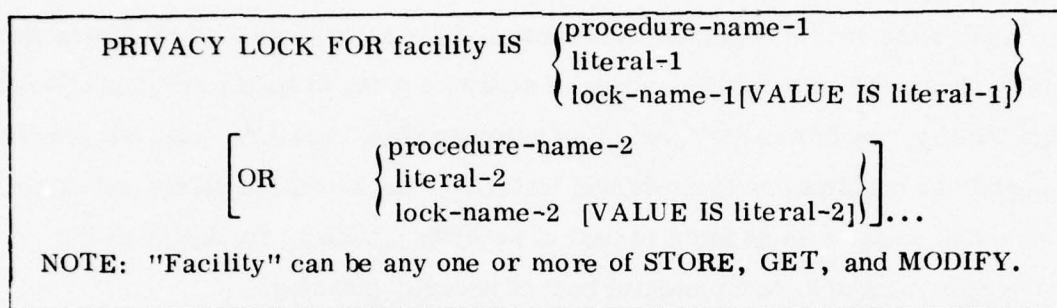


Figure 11-5. DBTG Implementation of Security [D]

The locks should be developed by the Data Base Administrator (DBA) for the Data Definition Language (DDL). Only the locks which are necessary for an individual process are given to the programmer for use in his program. The separation of responsibilities controls the dissemination of locks. However, there is no requirement for privacy controls; therefore, an improper design cannot be detected.

11.8.2 Integrated Data Store

Minimal security controls can be implemented in I-D-S applications by including an authority code number. A number, not to exceed 4095, can be assigned to a file and will result in a comparison of the given key with the defined key. The key is processed upon execution of the OPEN verb, as shown in Figure 11-6, and a successful match allows further processing. Failure of key matching inhibits file opening and returns an error condition.

* As defined by the DBTG

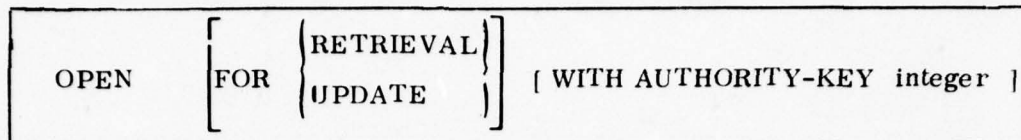


Figure 11-6. Implementation of Security in I-D-S [O]

11.8.3 World Wide Data Management System

WWDMS provides security control through the use of locks and keys. The facilities provided by WWDMS allow control of access to records and items by the user upon request for data. The PRIVACY time sharing subsystem compares the required lock versus the given key through a privacy file. The maintenance of the privacy file is the responsibility of the DBA through the time-sharing subsystem. The privacy file allows more than one key to access the same record/item and a key can open more than one lock. The lock can be associated with either read or write permission on the record/item. Figure 11-7 illustrates the clause used for the definition of records or items.

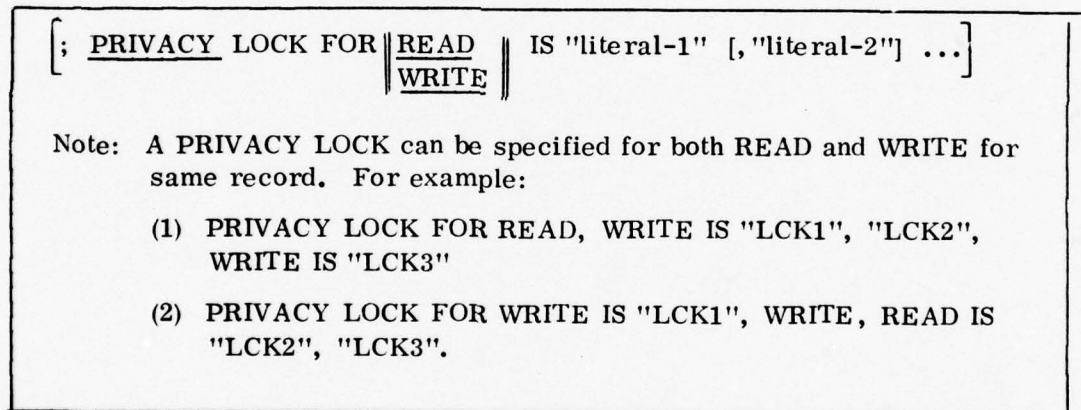


Figure 11-7. Privacy Clause [P]

CHAPTER 12 - NETWORKS AND DISTRIBUTED DATA

12.1 GENERAL

Time-sharing systems and remote job entry systems have been in existence for a number of years, allowing users to access a distant computer facility. More recently, computers have been interconnected to provide access to a number of facilities, increasing the scope of resources available to the user. Such a system permits the sharing of resources. A program residing on one computer is accessible from any other computer in the system. A data base may exist at one facility and be accessible to all, or it may be distributed throughout the system.

In addition to computer resources, a communication system is required to provide interconnection between the resources. A computer network, then, is more appropriately called a computer communication network.

12.2 NETWORKS

A computer network* is a set of interconnected computer systems which communicate with each other to allow resource sharing. In addition to the sharing of programs and data, load sharing is possible. Thus, a network can provide increased capabilities, economic savings, and increased reliability.

A computer network is actually a complex collection of many types of resources, including hardware, programs, data bases and operating systems. Any resource can be accessed from any other resource in the network. A program available to local users should be available to access the resources of any computer in the network.

The sharing of resources is achieved through computer-to-computer communications. Therefore, a communication system, sometimes called a subnetwork, is an integral part of a computer network.

*See Paragraph 12.4 for a brief description of selected network systems.

12.2.1 Components

Computer network components are classified as hardware, software, and protocols. Protocols are required to facilitate communication throughout the network.

12.2.1.1 Hardware

The principal hardware elements of a computer network are the host computers, communication switches, and the communication subnetwork.

12.2.1.1.1 Host Computer

The interconnected computers of a network are known as host computers, or simply hosts. The hosts may be all of one type (e.g., same basic hardware and operating under the same basic operating system), in which case the network is homogeneous. A heterogeneous network consists of a variety of host types. A host may provide service directly to a user or to another host.

A host is considered to be the complete computer system associated with a central processing unit (CPU). The system includes all peripheral equipment and may also include a variety of terminals, such as teletype, cathode ray tube (CRT), and remote job entry (RJE).

12.2.1.1.2 Communication Switch

Communication in a network is provided by a subnetwork. A host is interfaced with the subnetwork by a communications switch, which supports the basic communication services. Terminals (e.g., teletypes) in a network may be connected to a host or interfaced directly with the subnetwork through a special terminal switch. The switch functions may include segmenting, routing, flow control, and error control.

A logical unit of information to be transmitted through a network is a message. Some examples of messages are programs, data files, and telegrams. In many networks, messages are divided into packets for transmission. This segmentation can be performed by a switch or even a host. The switch forwards the packets independently to the destination switch, although, in some networks, complete messages are forwarded.

A switch is usually a minicomputer containing a number of buffers. The buffers are used to hold messages until they are forwarded for transmission, either to another switch or to a host. The number and size of the buffers depends upon the packet or message size, data rates and number of circuits. The amount of buffer space is generally kept to a minimum.

To efficiently utilize the circuits attached to a switch, a flow control technique is required. Flow control attempts to keep the buffers from becoming congested and maximizes the rate of information flow. Flow control protects a network from the sudden dispatch of a large number of messages to one destination. Buffers may be allocated at both the source and destination switches before transmission begins.

The network routing function is performed by the switches, usually based on some routing algorithm. As a message or packet travels through the network, each switch receiving it determines its next destination point.

Error control procedures, such as checksums, are used to assure the integrity of transmitted messages. Transmission errors may occur between switches or between a switch and a host. Therefore, procedures to detect, and possibly correct, such errors should be incorporated into each switch.

12.2.1.1.3 Communication Subnetwork

A communication subnetwork provides communication facilities among host computer and users. The subnetwork supports resource sharing and the transfer of information between hosts.

The physical path through which information flows is a circuit. The flow may be in one or both directions. The transmission may be either analog or digital, but analog is more common. However, in a network, digital data must be transferred

between hosts. Therefore, with an analog circuit, a modem (modulator/demodulator) is used at each end of the circuit. At the sender's end, the modem converts data from digital to analog form for transmission; at the receiver's end, the modem converts the analog information to digital form.

A channel is a logical path connecting hosts, or users and hosts. A channel may be distributed over several circuits, or a circuit may be multiplexed to support a number of channels. A channel may be dedicated for transmission between two points, but, more commonly, channels are shared.

Communication media can be classified as broadcast or point-to-point. Broadcast provides simultaneous transmission to, or reception from, a number of sources. An example of broadcast media is a radio channel. Point-to-point media provide transmission between two defined points, as in the telephone system. The majority of the computer networks in use, or under development, use point-to-point, rather than broadcast, media. Point-to-point transmission can be accomplished by circuit switching, message switching or packet switching. Finally, a communication switch may be required to perform one or more of the three types of switching; circuit switching, message switching and packet switching.

12.2.1.1.3.1 Circuit Switching

In a circuit-switched network, the source and destination are connected by a dedicated channel. The channel is established at the beginning of the connection and broken at the end, as in a telephone exchange. Communication may be transmitted between two hosts or between a user and a host. Circuit-switched networks generally use the public telephone system in conjunction with modems for converting data from digital to analog, or analog to digital, form.

The connection between a source and a destination often contains a number of switches. The switches perform channel switching to establish the communication path. The switching may be controlled by a minicomputer.

The primary function of the communication switch is to establish a connection and then to keep a channel dedicated to the connection until it is disestablished by users of the connection. Thus, circuit switching provides fast service and freedom for the user to use the channel as he wishes. Because of this freedom, the protocol between a host and a switch can be simplified. The primary disadvantage of circuit switching is that the line capacities are dedicated for each connection and subscribers may allow them to lay idle.

In circuit switching, channel allocation is based on the subscriber's priority and security requirements, but, once a channel has been allocated, it is kept dedicated for that connection until the connection is disestablished by the subscriber, or when abnormal conditions arise which require preemption of the connection by the operating system.

12.2.1.1.3.2 Message Switching

Message switching is a store-and-forward procedure. A message is transmitted through the network without dedicating a channel between the source and destination. Instead, each message contains a header specifying its destination address. The message switching system guides the message through the communication subnetwork to its destination.

A message switching subnetwork consists of point-to-point communication circuits and switching centers to interconnect the circuits. When a message arrives at a switch, it is stored in a buffer. The switch then selects the next switch to which the message should be routed to reach its destination. The message is forwarded to the next switch and the store-and-forward procedure is repeated, unless that switch is the final destination point.

In message switching, the source switch collects a full message from a source, establishes a connection with a destination switch, and sends the message thereto. Connections are assigned on a message basis. Even where a message is

divided into convenient size segments, as in AUTODIN I, the channel is dedicated over the entire period required to transmit the full message. All segments of a given message follow the same logical path from source to destination switch. They are not interleaved on a logical channel with segments of other messages. As a consequence, the proper ordering of segments of a segmented message is maintained throughout the network transfer process. Message switching protocols do not require that all segments of message be received by a switch prior to forwarding a received segment to the next switch in line, but since the accountability unit is the message, transmission error recovery procedures take into account an entire message at the source-to-destination switch protocol level.

The principal advantage of message switching is that the improved line utilization translates into greater effective net line capacity (effective line capacity less message switching overhead). Another advantage is that message segments always arrive at the destination switch in proper order. One disadvantage of message switching is that message segmentation does not eliminate the need for dynamic buffering because messages may vary in length. Other disadvantages are that the network is not transparent to the user, and speed of service is a function of the total load on the subnetwork. The user must provide addressing and security related information with every message. More work must be performed by the application programs in the switch to analyze message headers, to maintain message control blocks for each message in transmission, and to account for messages. Moreover, switching strategy must take into consideration this extra work. Finally, the dynamic buffering scheme must be sufficiently complex to avoid unduly fragmenting the total storage space.

12.2.1.1.3.3 Packet Switching

With packet switching, a user's message is partitioned into a number of segments called packets. Each packet is of a predetermined maximum size and is independently transmitted from source to destination. The transmission is a store-and-forward procedure, as in message switching.

Each packet contains address information and a sequence number. If packets arrive at their destinations, possibly out of order, reassembly is required to put them back into proper order.

In packet switching, a source switch receives data from a subscriber in fixed length packets. The source switch does not wait for the reception of a full message. Instead, when it receives a packet, it starts forwarding the data to the destination switch. Since a packet is the unit of transmission, each packet requires a packet header identifying the source, destination, security, precedence and other fields for accountability purposes.

A packet length is usually small compared to the message. Packets originating from various subscribers, but destined for the same switch, are interleaved on the line from the source to destination switch. Thus, the line capabilities can be utilized with greater efficiency than in the other two types of switching. Because the buffer space is the same for each packet, it becomes possible to simplify the buffer management scheme and achieve high throughput with low overhead.

The primary disadvantage of packet switching is that the network is not as transparent to the subscriber. The user must provide addressing, security, and error detection related information with each packet. In general, the resource management for a packet switching environment is more complex than that for the other types of switching.

The problem of resource management is even more complex for switches that must support more than one type of switching. In such cases, there are ways to adaptively allocate a channel, both for packet switching and circuit switching such that to a circuit switching function it will look like a dedicated channel, while in fact the data is actually packetized. Such a scheme, however, imposes complex requirements on I/O handlers so that they can cleverly multiplex the two types of switching. Similarly, the buffer management scheme should be able to manage storage space to support more than one type of switching system.

12.2.1.2 Software

The computer network software performs the functions of network control, message queueing, message editing, and error recovery. This communication software is in addition to the operating systems and application programs residing in network hosts.

12.2.1.2.1 Network Controller

The network controller is the software which provides the interface between network hosts and the communication subnetwork. The subnetwork is responsible for the transmission of messages from host to host. The network controller must manage buffers used for storing messages (or packets) during transmission. It must record statistics, such as transmission delays and error rates, which can be used in monitoring network performance.

In controlling host to host message transfer, the network controller manages the transfer of messages between a switch and a host.

12.2.1.2.2 Message Queueing

In a network, it may be necessary to temporarily store messages until required resources are made available. Message queues are used for this purpose. The queues contain messages received in the subnetwork but not yet delivered to the appropriate host and messages generated by a host but not yet delivered to the subnetwork.

The message queueing controller queues messages by priority, which may depend on order of arrival, priority assigned to the source, or information in the message header. The queues may reside in main or secondary storage. Secondary storage is desirable if messages are to be held for later transmission.

The message queueing controller may also initiate activation of an application program when a message requires processing by the program.

12.2.1.2.3 Message Editing

The message editor transforms outgoing messages into a standard form for transmission and incoming messages into a form acceptable by the receiving host. Messages usually require headers for transmission. The message editor must insert and remove headers at appropriate times (i.e., before and after transmission).

12.2.1.2.4 Error Recovery

Error recovery functions are required in network software if the integrity of the network is to be maintained. A failure can occur at any point in a communication subnetwork* and procedures must be available to detect and recover from these failures.

Mechanical and electrical faults may occur in the transmission lines or in any hardware device in the subnetwork. Such faults usually produce incorrect checksums or timeouts.** An incorrect checksum causes a message to be retransmitted; a timeout may cause a device to be logically disconnected.

Faults may occur in the storage used for buffers and message queues. To prevent loss of information, messages should be recorded on a journal (Paragraph 10.3.2) for recovery purposes.

Failures in a subnetwork may also occur in the software. To decrease the likelihood of such faults, consistency checks and recovery procedures should be incorporated within the software. Whether caused by hardware or software, each fault should be recorded for evaluation and analysis. In addition, test and diagnostic routines can be run periodically and should be run during normal network operation.

12.2.1.2.5 Data Traffic Type

A switch may be serving users with varying types of data transfer needs, and each type of data may impose some requirements on the associated operating system. Although other types of data may pass through a switch, three types are of particular interest: interactive, narrative, and voice (stream).

* See Paragraph 12.2.1.1.3.

** A timeout occurs when an expected message or reply is not received within a specified time interval.

Interactive data must traverse the network with minimal delay and high reliability. The switch must establish the connection with utmost speed and transfer interactive data at a rate such that the network and the switch do not impose undue time delay. Generally, interactive traffic volume is small; thus, it is anticipated that this type of data will not require large throughput rate.

Delay is not so critical in narrative (or bulk transfer) traffic, as it is in interactive traffic. However, reliability is of equal importance. In addition, narrative traffic calls for high throughput rate since the data volume is usually large. The switch must meet this high throughput requirement; otherwise, the traffic may back up, and end-to-end delay may become prohibitively large.

The third type of traffic, voice (stream) must be supported with minimum delay. An equally stringent requirement is for high throughput since the data volume is usually large. There is some relief in that voice traffic is highly redundant and consequently high reliability is not essential as in the other two types.

These three types of traffic are shown in Figure 12-1. Each side of the triangle represents an axis with a characteristic such as low delay. Each of the three corners identifies a traffic type which requires the characteristics shown by the two edges meeting at the corner and the less important third characteristic indicated by the third side. For example, corner C represents voice traffic and indicates that high throughput and low delay are essential, while high reliability is less important. Various tradeoffs can be made to arrive at resource allocation techniques to support more than one type of data traffic at the same time.

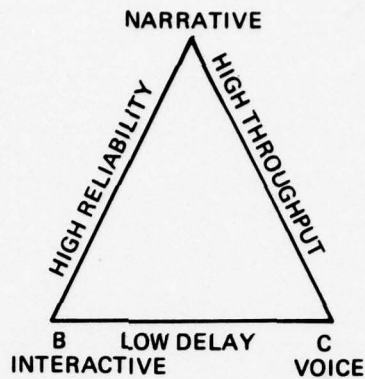


Figure 12-1. Data Traffic Types

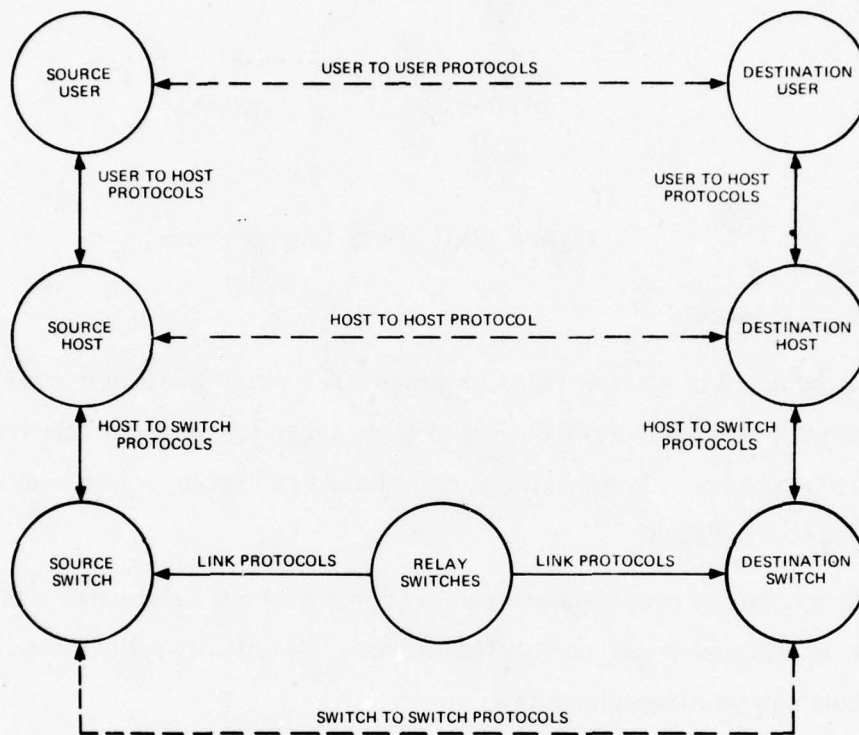
12.2.1.3 Protocols

A protocol is a set of rules or procedures which facilitates communication throughout a network. Protocols specify message formats and relative timings of message exchange. By adhering to established protocols, a new host can more easily be added to a network.

A number of protocols are required in a network to transfer a message from a user on one host to one on a different host. Practically all common networking protocols can be categorized as a:

- User to user protocol
- Host to host protocol
- Switch to switch protocol
- User to host protocol
- Host to switch protocol
- Link protocol.

Figure 12-2 illustrates the interrelation between protocol categories.



NOTES:

DASHED LINES BETWEEN NODES DENOTE FUNCTIONS IMPLEMENTED THROUGH LOGICAL INTERFACING APART FROM ACTUAL MOVEMENT OF DATA.

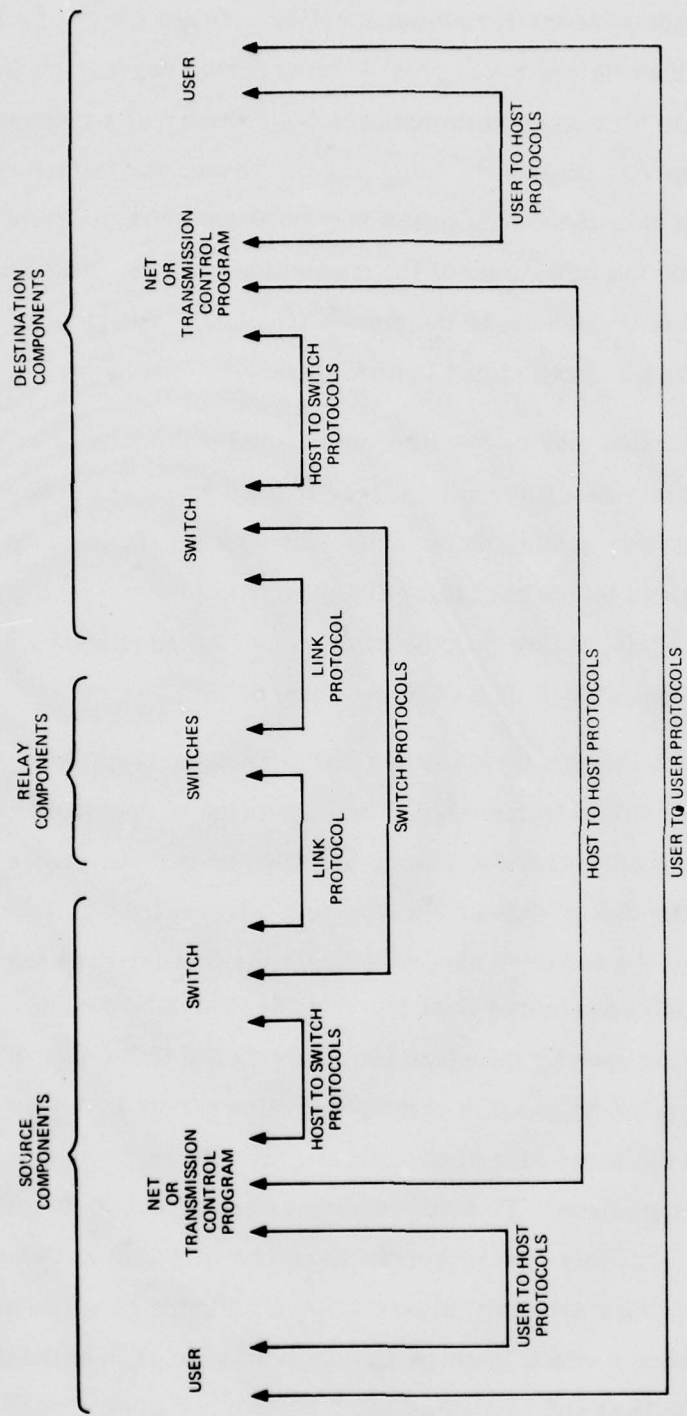
SOLID LINES INDICATE FUNCTIONS PERFORMED IN LINE WITH ACTUAL MOVEMENT OF DATA.

Figure 12-2. Protocol Categories

Protocols provide a means for communicating between nodes of a network. Interfaces between different protocols provide an orderly progression from one protocol to another. This ultimately culminates in the delivery of a message to a pending receiver or an appropriate diagnostic indicating the reason for failure to do so. A given protocol on one side of the connection may perform quite different functions than its counterpart on the other side of the connection. Also, some protocols (e.g., host to switch) affect only one side of the connection, while others (e.g., host to host) are initiated in one host and completed in another.

A particular function may be required by several protocols. For example, a flow control function may be required by a user to host protocol to prevent monopolizing network operating system buffer capability by a single user. A similar function may be required by the host to switch protocols to avoid losing message data on the switch. Again, a flow control function may be required by a switch protocol to facilitate transfer of data over the subnetwork.

Communication between network nodes at the protocol level is generally accomplished via appendages added to message data. Layering of protocols generally refers to the order in which the information associated with the various protocols are appended to and deleted from the unit of data under transfer. Generally, the layers are assigned symmetrically so that the software associated with the last layer on the source side of the subnetwork is acted upon by the first layer of the destination side. Appendages are developed and acted upon by interface protocols on the same side of the subnetwork, but, here again, the same symmetric principle applies except that the appendages are added and deleted on the same side of the subnetwork. Appendages are deleted by the protocol which acts upon them. To avoid forcing awkward and/or inefficient implementation of networking protocols, the protocols should be mutually exclusive, when one is being acted upon the others are not. Figure 12-3 and Figure 12-4 illustrates the layering concept of a user message which is small enough to be sent across the network as a single unit. User-To-Host and Host-To-Switch protocols can be described as interface protocols in that their action is initiated and completed on the same side of the subnetwork. In Figure 12-4, each item reflects the state of the data and data appendage upon completion of the respective protocol.



NOTE: "BUCKETS" DENOTE EFFECTIVE RANGE FOR PROTOCOL DESCRIBED. THEY REPRESENT PROCESSING INTERVAL OVER WHICH INFORMATION (i.e. ADDRESSING, ROUTING, ETC.) FOR EACH RESPECTIVE PROTOCOL MUST BE PRESERVED.

Figure 12-3. Protocol Layering

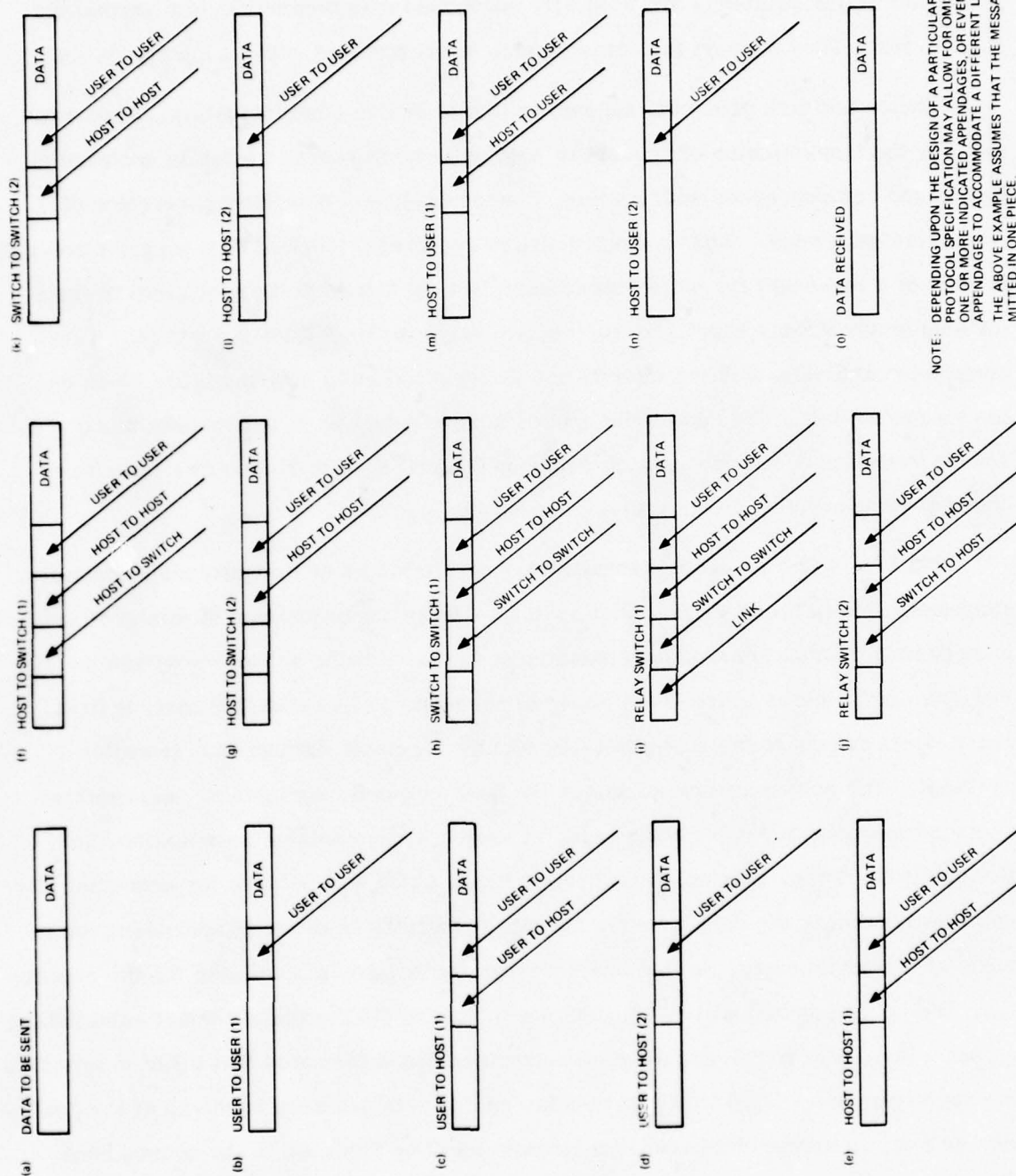


Figure 12-4. Protocol Appendage Cycle

12.2.1.3.1 Low Layer Protocols

Low Layer protocols are basically concerned with communication across the subnetwork. They support the transmission of bit streams without interpretation.

Switch and link protocols support switch to switch communication. They provide for the transmission of packets or segments from source switch to destination switch and are concerned with routing, flow control, and detection/correction of transmission errors. When a source switch receives a block of data (e.g., a message or part of a message) from the source host, it may fragment the block into packets (or segments) of convenient size for transmission to the destination switch. This conversion and other related information is recorded in an accountability block on the source switch. The destination switch acknowledges each unit of data using source switch to destination switch protocol (Figure 12-3). The source switch records such acknowledgements into the accounting block.

Host to switch protocols (sometimes referred to as access protocols) permit communication between a host and a switch. The primary function of this protocol is to permit information exchange pertaining to data flowing between host and switch. For example, when a source host wants to send data to a destination host, it first must obtain access to the subnetwork by identifying itself through host to switch protocol. The source switch examines the host supplied information. Recognizing a new connection, it checks addresses, precedence, and related information, and, if there is no error, sets up an accounting block which will be used for accountability and flow control in the data transfer phase. If there is an error in addressing or in some of the parameters, or if subnetwork resources are not available for the connection, the source switch will notify the source host of the reason(s) for not establishing a connection. Furthermore, when an accounting block indicates that all of the packets (or segments) associated with a particular unit of data has been received at the destination switch, an appropriate acknowledgement must be returned to the source host. The host to switch protocol performs these functions. Similarly, the destination

switch and destination host communicate through host to switch protocol. Finally, the host to switch protocol is used to terminate subnetwork connections.

12.2.1.3.2 High Layer Protocols

High layer protocols are concerned with the interpretation of data. They provide such functions as interactive terminal support, batch service support, file transfer, and work load sharing. The high layer protocols support the general network user and specific network applications.

Host to host and user to host protocols provide for communication between a user and the network. These protocols are controlled by the host system and are concerned with the movement of data between the user and the subnetwork. The host to host protocol establishes the conventions for information exchange between hosts; in a heterogeneous network, the host to host protocol must handle the differences between host systems (e.g., byte and word structures, file formats). User to host protocol allows users to establish and terminate logical connections within the network. It provides the means for network users to send and receive data and interrupt signals over the network. Host to host and user to host protocols must be implemented on every host to provide local user access to the network and remote user access to resources on a particular host.

Examples of high layer protocols are:

- Remote Interactive Terminal
- File Transfer
- Remote Job Entry (RJE)
- Teleconferencing.

12.2.1.3.2.1 Remote Interactive Terminal

The Remote Interactive Terminal protocol allows a user at a terminal to communicate with a remote host and appear to the host to be a local terminal. The user operates from his local host and need not know the characteristics of the remote, or

server, host. The protocol handles the differences of character code, timing, and processing of interrupt signals. The user host converts all characters from their internal representation to a standard code for transmission; the server host converts from the standard code to its own internal representation. The Remote Interactive Terminal protocol can support timesharing and the direct access* mode of operation with a remote host.

12.2.1.3.2.2 File Transfer (FTP)

The FTP supports the transfer of files between network hosts; hence, it provides for the sharing of programs and data. In general, a file in one host may be moved to any other host in the network by a user located at yet a third host. The FTP uses lower level protocols for file transmission. These protocols handle conversion of files into data streams for transmission and the reconstruction of the files in proper format at their destination.

12.2.1.3.2.3 Remote Job Entry (RJE)

The RJE, or remote batch computing, protocol allows a batch job to be submitted from any appropriate device and executed on any host in the network. This protocol uses the FTP for the submission of jobs and the transmission of output. A user must not assume that a job set up for one host will run on any other, particularly in a heterogeneous network. For example, the Job Control Language (JCL) may differ on the hosts and the RJE protocol cannot presently handle such a situation.

12.2.1.3.2.4 Teleconferencing

Teleconferencing provides a conference capability to terminal users, and in conjunction with Remote Interactive Terminal Protocol, serves both local and remote users. The structure of the conference consists of a chairman and many participants. Basic features, available to all participants are:

- Send messages to other participants
- Connection to other direct access programs (such as timesharing) or a user program

*Honeywell terminology

- Review previously entered conference messages
- Receive printed copies of conference minutes.

12.2.1.3.3 Network Control Language (NCL)

NCL allows users to integrate a sequence of operations to form a cohesive processing unit in much the same way as the operating system combines separate user computing activities into a single job unit. NCL statements allow users on any host to:

- Transfer a data file from one host to another
- Transfer a job from one host to any other host for execution
- Initiate workload sharing activity between hosts.

Furthermore, an NCL provides users with control over their respective network executions through a set of control statements which provide:

- Status information
- Cancellation directives
- Conditional control over execution of individual statements.
- Recovery options which can be invoked to minimize the overhead associated with recovery from unscheduled system failures.

12.2.2 Organization

The organization of a computer network may be classified as:

- Centralized
- Ring
- Distributed
- Completely connected.

12.2.2.1 Centralized Network

A centralized network contains a central computer (host) which services all other hosts in the network. All communication lines are connected to the central computer and all host to host communication is transmitted through the central com-

puter. A disadvantage of the centralized network is that a failure of the central host will disrupt all communications.

A centralized network, also known as a star configuration, is illustrated in Figure 12-5(a).

12.2.2.2 Ring Network

A ring network is a circular operating organization of hosts. Each host is connected to the communication ring by a ring interface. Transmission through this ring is generally in one direction only. A message sent by one host travels around the ring until it reaches its destination. Intermediate hosts ignore the message.

A ring network may transmit entire messages or data units. No routing decisions are required during transmission. In a ring network, a failure of one host to host link will disconnect the network unless redundancy has been implemented.

A ring network is illustrated in Figure 12-5 (b). For simplicity, the ring interfaces are not shown.

12.2.2.3 Distributed Network

In a distributed network, major hosts are interconnected. Other hosts may be connected to any major host. Each pair of hosts in the network is connected either directly or indirectly through intermediate hosts. There are at least two paths between any two host. Intercomputer Message Processors (IMPs) in the network perform the routing function.

The distributed network is the most common organization presently in use. Its cost is less than the completely connected network and it is more reliable than the ring and centralized networks.

A distributed network is illustrated in Figure 12-5 (c). The switches are not shown.

12.2.2.4 Completely Connected Network

A completely connected network contains a communication link for each pair of hosts. If one link fails, message (or packet) transmission is possible by another route. Because of the number of connections required, the cost of a completely connected network is very high.

Figure 12-5 (d) illustrates a completely connected network. Although not shown, an interface is required for each host. A completely connected network is a special case of a distributed network.

12.3 DISTRIBUTED DATA BASE

12.3.1 General

A distributed data base (DDB) is a scheme whereby files or data utilized by one or more processors are dispersed among other processors, as opposed to data which is stored in a computer file or data base at a centralized processing site. A DDB is composed of at least one, but usually multiple, logical files or data bases. A data base may be distributed on the basis of ownership, usage, origin, subject, content, maintenance, geography, etcetera. To carry the concept further, a data base may be thought of as a logical structure which comes into being whenever some operation or processing is performed on or with data. Consequently, data can be dispersed throughout the network and need not be confined in a formal or physical file. It is important to know how the data elements are related to each other and how they are distributed throughout the network.

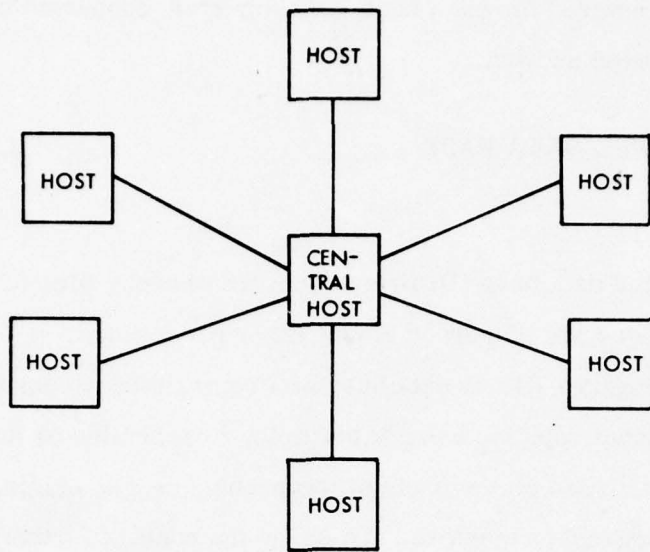
12.3.2 Environment

The DDB concept is particularly adaptable to a computer network environment. This network was discussed in Paragraph 12.2; however, in this discussion, it will consist of two or more data processors and network processors with permanent files appended, in addition to other hardware usually associated with a computer network (i. e. , communications, communication processors, terminals, etc).

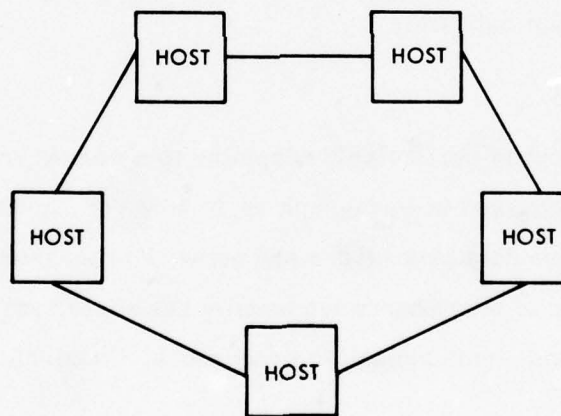
12.3.3 Concept

A DDB exists when an intercomputer network includes permanent files or data residing in one or more nodes of the network, and these files or data are available to all network users with approved access. The DDB is applicable for use when:

1. Users of data are located at many different sites.
2. Data is located at many different sites.

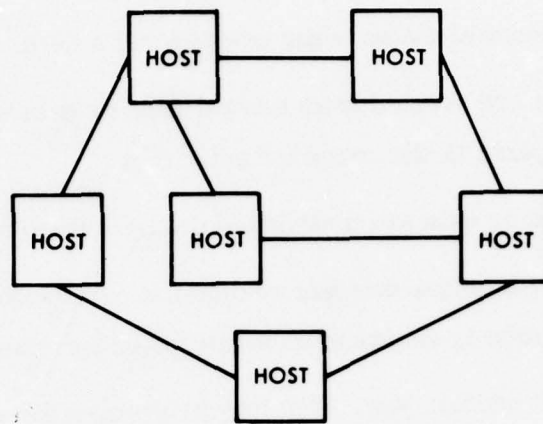


(a) CENTRALIZED (STAR) NETWORK

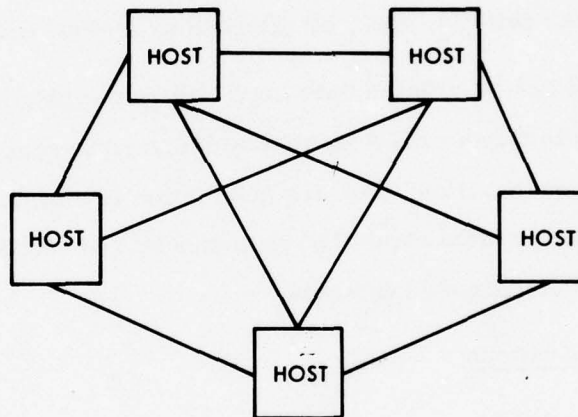


(b) RING NETWORK

Figure 12-5. Network Organization (1 of 2)



(c) DISTRIBUTED NETWORK



(d) COMPLETELY CONNECTED NETWORK

Figure 12-5. Network Organization (2 of 2)

12.3.4 Creating a Distributed Data Base

There are several methods for establishing a DDB:

1. Subsets are created from central files or data which are sent out to other participants in the computer network
2. All files or data are distributed throughout the network
3. Files with unique data are resident at certain nodes of the network and information is accessed when needed by the other members of the network.

Looking at it another way, files may be grouped for various reasons such as geography, type (e.g. subject matter), primary updating responsibility, proximity to source of data, etcetera. Each group may be assigned to a particular information processor, duplicated and attached to each of the data processing computers, split because of size, activity, etc., or attached to several processors.

The DDB can be created both implicitly and explicitly [Z]. If the data base has been generated from files created by individual users, it is said to be implicit. As each user creates files, they are automatically distributed in the processor he uses. DDBs are created explicitly when special care is taken to disperse an integrated file into distributed segments.

12.3.5 Considerations

12.3.5.1 Geography

Geographic distribution usually is a function of usage or transmission cost of data to and from a processing center. It is common to place the data file in a computer located at the site of prime or most usage. This reduces the cost of transmitting the data, since shorter transmission lines mean lower costs.

Once a given file is created, other organizations may use the file. This usage may grow to where the communication costs will force a copy of the file to be placed at various locations to reduce the communication costs.

12.3.5.2 Type

Data files are often located where there is direct subject matter responsibility, or the responsibility may extend to several files containing a particular type of data, but possibly located at different sites. Grouping of files by type may be closely associated with placing the file at the source of the data, e.g., all files containing personnel data may be located at a personnel center, or financial files at a finance center.

12.3.5.3 Primary Maintenance Responsibility

It may be expedient to locate a file or data in a computer data base close to the responsible maintenance organization, since it is assumed that an organization which has a primary subject matter interest and updating responsibility will maintain the data in as current a state as possible. The data will probably be more current than data at remote sites to which it must be sent via telecommunications, mail, or courier. However, this concept, carried to the extreme, may reinforce the argument for a single copy data base rather than a DDB. This argument becomes especially acute when considering data bases and their alternates which must be maintained in real-time.

12.3.5.4 File Distribution

When discussing DDBs, one is not limited to multiple copies of entire data bases distributed as per some algorithm. Distribution of file segments must also be considered. That is, certain files may be structured, and their usage may permit distribution at the file segment level. Thus, in many instances, the possibility that many relatively small data base segments are being distributed must be considered. The impact of this type of distribution upon directories and maintenance organizations is substantial.

12.3.6 File Segmentation

Splitting or segmenting of logical files is accomplished by attaching parts or segments of a file to different information processors. There are several ways to handle a segmented file |W|:

- A single copy of each data segment
- A single primary copy of each segment, with passive backup copies
- A single primary copy of each segment, with active backup copies
- A single primary copy of each segment, with partial backup copies
- Multiple primary copies of each segment.

12.3.6.1 Single Copy of Each Data Segment (Figure 12-6)

Only a single copy of each segment is updated, but these may be located in different host computers. Access to the data in a particular segment may be through the network if the segment is not residing locally. It is apparent that, in some cases, most of the maintenance or inquiries are via the network. For those segments which contain frequently accessed data, processor loads can become extensive. However, in most cases, the processor activity will not be greater than when a nondistributed data base is accessed.

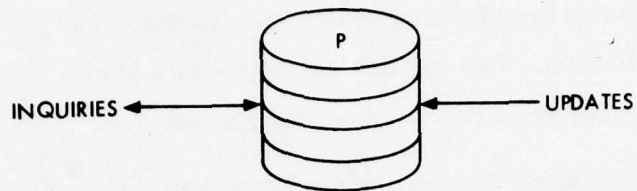
Response time to queries is a function of network availability and the processing done by the segment owner. Since the segment is residing in only one host, its availability is limited by the processing load on that host.

Processing loads may increase slightly for the querying host because of the additional directory processing required to locate a wanted segment. The same may be true for storage, since the directory must include host location criteria.

There are some drawbacks to this scheme. Reliability is low since there is only one copy of a given segment, and loss would make the segment unavailable to a user. In addition, network delays increase the time for remotely accessing a segment.

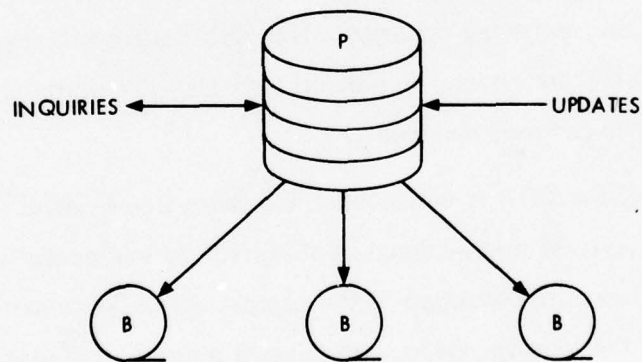
12.3.6.2 Single Primary Copy with Passive Backup Copies (Figure 12-7)

This process, also known as remote journaling, provides that a journal of update transactions for a segment located at the host of data responsibility is kept on other remote hosts with a copy of the segment. Each update generates a message



P = PRIMARY COPY

Figure 12-6. Single Copy



P = PRIMARY COPY; B = BACKUP COPY

Figure 12-7. Single Copy With Passive Backup

for each remote journal. However, if necessary, these journal updates can be batched and sent as a unit, thus reducing network traffic.

Keeping the journal may impose some additional load on the data processing computer, but increased reliability makes this added load tolerable. If the primary segment or host should become disabled or inaccessible, the remote journal will become a defacto backup. The old copy on the remote site(s) is updated by the journal and, consequently, a new primary copy is created.

Storage requirements for the remote journal will be small and inquiry traffic and query response time will be unchanged over a single copy. The only change in the case of update over a single copy is that a remote journal entry must be created.

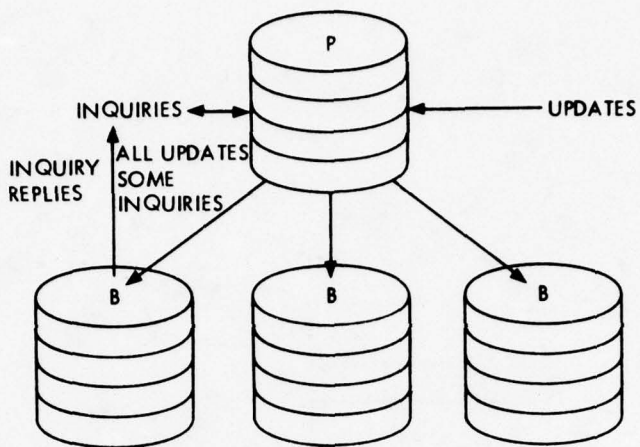
12.3.6.3 Single Primary Copy with Active Backup Copies (Figure 12-8)

This scheme uses a primary copy and backup copies which are maintained in a state as current as the primary segment. The owner of the primary copy has the responsibility to keep the copies updated at the same level as the primary segment. Queries may be directed either to the primary segment only, or they may be directed to any segment (i.e., primary or copy). Network traffic will depend upon the number of updates, number of inquiries, how these inquiries are distributed, and the location of the copies and/or primary segment.

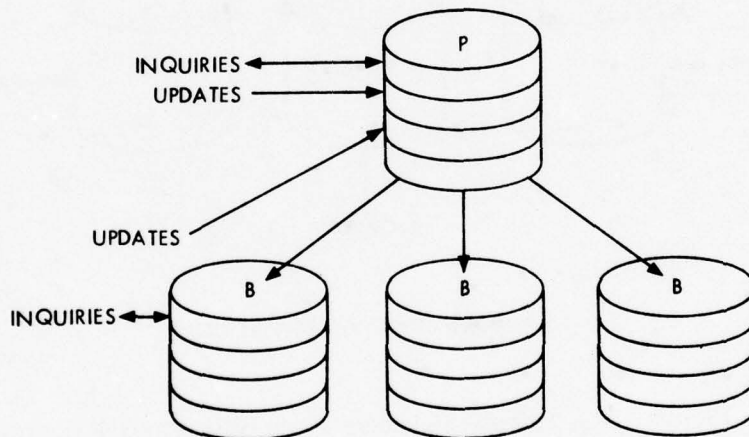
Reliability of the DDB is enhanced since there are several active backup copies, each at the same level of maintenance as the primary segment. Response time to queries is a function of the location of the copies and the primary segment, origin and destination of the inquiry, processor load at a particular site, and existing network traffic. However, the problems of maintaining consistency between primary and numerous backup copies can be considerable if operational exigencies require consistency over relatively short time frames.

12.3.6.4 Single Primary Copy with Partial Copies (Figure 12-9)

Under this concept, a single primary segment is maintained at a site and partial copies are distributed via the network after each update. Whereas the primary segment contains all data, the partial copies contain only the data of primary interest



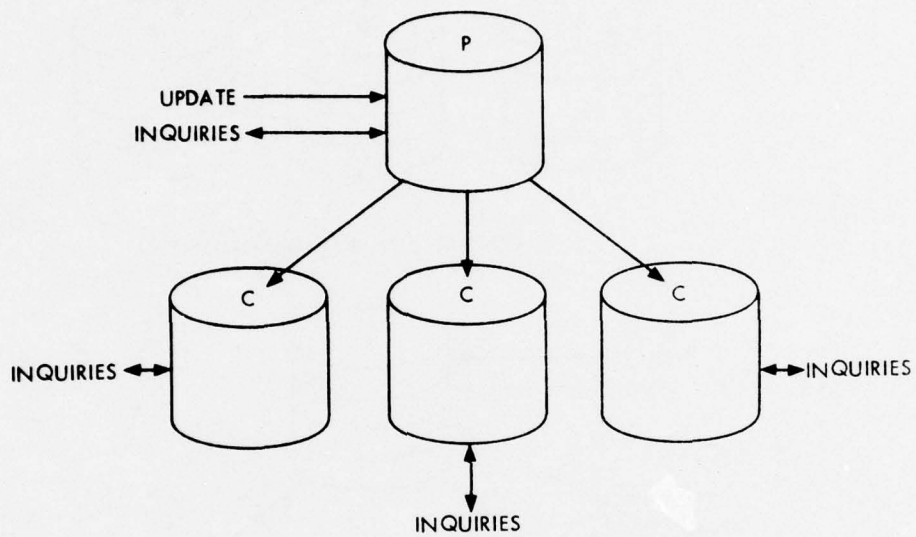
A. CENTRAL INQUIRY MANAGEMENT



B. DISTRIBUTED INQUIRY MANAGEMENT

P = PRIMARY COPY; B = BACKUP COPY

Figure 12-8. Single Copy With Active Backup



P = PRIMARY COPY; C = PARTIAL COPY

Figure 12-9. Single Primary Copy With Partial Copies

to the organization located at each site. Inquiries may be directed to any copy, but responses are limited to the data contained in the partial copy. All other inquiries must be directed to either the copy which contains the desired information, or to the primary segment.

Updating is performed on the primary segment only, and updated copies are distributed after each update. By this means, copies are always as current as the primary segment, but data can only be changed by the owner of the primary segment. Network traffic during prime time is reduced since inquiries are usually directed to the nearest copy. Requests for data not contained in the partial copy which must be sent to the host of the primary segment are rare. Storage is reduced since only partial copies are located at each site outside the primary segment site.

Reliability is enhanced since, besides the data being stored in the primary segment, the same data is distributed throughout the other sites which collectively contain the same data.

As one considers the variations of DDBs, additional configurations can be developed which are feasible and may be the best solution for a given network situation. One of these configurations is that of multiple primary copies of data segments.

12.3.6.5 Multiple Primary Copies of Data Segments (Figure 12-10)

Each segment is autonomous, i.e., no single host controls the updating or inquiry functions. Inquiries and updates may be directed to any segment; however, all updates of the data files in one host must be coordinated with the holders of the other copies of the segment. All updates must be synchronized between all copies, which will increase network traffic considerably during update periods. Since network traffic may be heavy during updates, inquiries can be delayed until the update is completed.

One of the problems associated with segmented files is how to catalog such split files. A separate name can be given to each segment or the segments can be given the same name in the catalog. Sophisticated software is required to handle a logical or physical file segmentation.

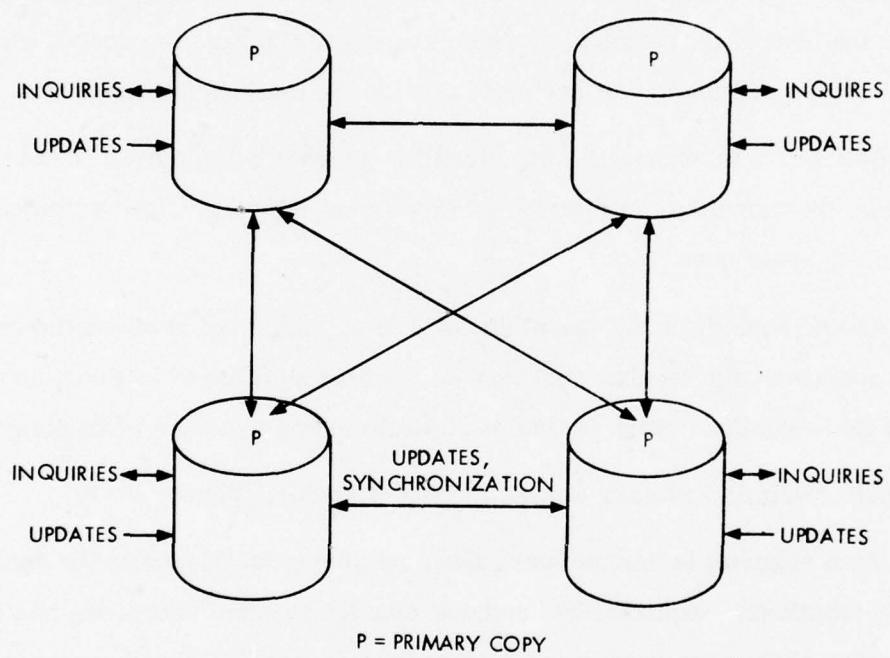


Figure 12-10. Multiple Primary Copies

To resolve this problem, the concept of distributed directories has been advanced.

12.3.6.6 Distributed Directories

To carry the concept of distributed data one step further, instead of distributing formally organized or segmented files, the information about the data may be distributed throughout the network. In large systems with many on-line users, data files, and processors, a directory of some type may be required to steer the user to the data he requires.

An automated directory may be simply a description of the relationships between data bases, files, data sets, records, and fields. Other levels of directories are possible, such as record directories showing the blocks of information within records, and device oriented directories specifying the contents of devices, i.e., the location of files, records, blocks, fields. A directory may be expanded to provide space for audit trails, last update, status, and secondary passwords.

An additional refinement will improve the directory capability by incorporating a thesaurus to aid the user in his directory search. Such a thesaurus would contain all colloquialisms, synonyms, nomenclature, and keywords required to identify the host and site of a particular data item. A Boolean capability along with the thesaurus will allow the user to combine thesaurus terms to locate more precisely the desired network resources.

The directory concept may also be used to identify, locate, and connect to other network users for teleconferences.

In general, network directories include:

- A listing of all resources pertinent to the community of network users
- An identification function to assist users in identifying required resources and owners of such resources
- A location function to locate identified resources and provide the information required to address these resources
- A security function to prevent unauthorized access to directory information and services

- A maintenance function to provide network management with adequate dynamic control over directory operation to assure fail-soft characteristics during periods of unusual stress
- A statistics function to provide network management with statistical information to optimize directory and overall system performance
- A user-assistance function to provide on-line tutorial help to enable directory users to make optimal use of the distributed directory
- A directory language to allow all users to communicate with the directory
- A resource description language to allow all owners of network resources to describe them to the directory.

12.3.7 DDB File Access Methods

Once it is determined that a logical file can be allocated between two or more processors, the multiplicity of file access mechanisms must be investigated.

Different requirements may include access to file(s) [BB]:

- Connected to and accessed from a single processor
- Updated at one location, but retrieved from many locations
- Retrieved from one location, but updated from many location(s)
- Retrieved and updated from many locations.

With respect to the second, third, and fourth items, access methods which are satisfactory for small volumes may be completely unacceptable for large volumes.

12.3.7.1 Single Processor Files

The first type of access represents standard data processing, is simple, and widely understood; therefore, the remainder of this paragraph describes possible ways of providing the other types of access.

12.3.7.2 Complete File Transmission

Simple inquiry/response access to files does not normally require complete file transmission. The break-even point at which access volume makes it less costly to send the entire file than to send individual items should be determined for each file type and utilized to determine the proper access method for a given file.

A complete copy of a file can be transmitted from its present location to another processor, so that it can be used by a job executing there. The decision to do this can be made either automatically by the operating system, or manually by a user with the file access requirement. Some obvious restrictions must be placed on this network resource. First, some limit should be considered on the size of files which can be transmitted in this way. Second, if the user wishes to update the file copy which is transmitted to the remote location, care must be taken to prevent interference with any updates in the primary location. This can be done by preventing data base changes at the primary processor until an updated copy is returned from the other location, assuming that the remote update has precedence.

Transmission of a complete file copy has the value of simplicity. Once the file is received by the other processor, it can be cataloged normally, and the same access procedures used for all resident files can be applied to it. Depending on the number of accesses required, this method may also reduce transmission time and overhead. Sending the entire file as a block is clearly more efficient than sending it record by record in response to individual requests. This technique allows access to multiple files for either inquiry or update. Update is permitted as long as access requirements for the file allow it to be locked while the remote update is in process.

12.3.7.3 Operating System Control

The processor's operating system can provide automatic access to file data at other locations. The sequence of events to handle such an access might be:

- User program issues request for file data
- Operating system discovers that file is attached to another processor
- An I/O request is sent to appropriate processor
- Other processor receives request, processes it, and returns either requested data or a denial.

To avoid delay and interference, retrieve only is normally allowed, but the file may be modified by other jobs during retrieval. Data retrieved from multiple records in the file may therefore be inconsistent, due to concurrent updates taking place. This retrieve only restriction is made because other access modes (update, exclusive retrieval, etc.) have the following potential problems:

- Jobs in each processor must wait for data to be returned from other locations. In the worst case, these jobs (although inactive) might tie up enough system resources to hamper operation. If multiple concurrent update users of the same file are executing at different locations, extensive delays might occur. Deadly embrace conditions could be caused between jobs; each of two jobs could have files locked at a different location, and the two would be mutually waiting for access to the other's files.
- Also to be considered is the complicated software required, with its resulting overhead, to control the multiple update and recovery aspects. An extremely complicated mechanism is required to recognize dependencies between jobs, detect and resolve deadly embraces, and control recovery of an abnormally terminated job which was updating files at multiple locations.

To prevent misuse, accidental or intentional, of this mode, the user may also be required to specify, at time of file access request, the expected number of accesses.

If none is specified, a default limit should be applied. If accesses are attempted in excess of the limit, the job should be terminated if not sanctioned for the level of activity encountered.

The concept that the operating system should provide automatic access to file data at other locations is attractive. Its most obvious advantage is simplicity from the user's point of view. In this mode, the user need not worry about file separation or file location, as long as he requires only retrieve mode access. An advantage as compared to complete file transmission is that only data actually accessed need be transmitted. If the access volume is below the point where transmission of the entire file becomes economical, or the file is too large to send, this mode can save transmission time and cost. The major disadvantage is the complexity of the software required to support this mode. All necessary logic must be contained within the software system. This type of access seems feasible only where a limited number of remote inquiries is required. In other cases, the disadvantages very likely outweigh the advantages.

12.3.7.4 Job Chaining

The following method can be used when one application requires access, particularly update access, to files attached to more than one processor. The job which would otherwise update multiple files at multiple locations can be rearranged as a series of jobs chained together by means of a job activating scheme. Each job in such a chain would update only the file in a specific processor. As each job finishes, it activates the next job in the chain, generally in a different processor. This method may require that the analyst and/or programmer be aware of the exact file distribution among processors or alternatively that the data base directory chain the update job from processor to processor.

Using this technique, no limits need be placed on access type. Since each job execution at each location is independent of the others, except for the passing of control, the problems of delay and interference noted under the previous method should not occur.

The advantage of using job chaining to accomplish unlimited access in multiple processors is that it provides great flexibility without apparently causing any interference with other update users. From the operating system's point of view, it is extremely simple, since all complex logic becomes the user's problem. However, there are several disadvantages to this approach. One is that the application designers/programmers may have to be aware of file distribution and take this into account in system design. Second, each job in a chain is independent of its predecessors, once it is in execution. If an abnormal termination occurs somewhere in the chain, breaking the chain, it is difficult to remove the effects of preceding jobs. Of course, the termination notice resulting from a chained job can optionally be sent to the originator of the first job, who can then take appropriate action. This type of job chain requires careful design, so that a task can be re-initiated at any point if an abnormal termination occurs. Finally, there appears to be no way to present an unchanging picture of data base status. The data retrieved by each job represents that part of the data base at a different point in time than the data retrieved by the other jobs. Thus, extensive time stamping may be necessary. In contrast, an exclusive access job running in a single processor can obtain a completely static picture of the entire data base because no other job has access to the file.

12.3.7.5 Delayed Updating

Another mechanism which can be used to provide access to files at multiple locations is delayed updating. Delayed updating allows multiple jobs to concurrently retrieve from an apparently unchanging file. Changes made by each job are posted, not to the main file, but to a change file. These change files are inaccessible to any other jobs. Since changes are inaccessible until they are applied to the main file, they cannot affect other jobs in concurrent execution. Delayed update changes can be applied to the main file when each job finishes execution successfully, or by some other algorithm. Multiple delayed update jobs can be run concurrently if there is a program available which can synchronize concurrent changes to the same record within the file. Such a synchronization program must, of course, be tailored to the

format and conventions of the file type (indexed sequential, inverted, linked, etc.), and thus may require access to the definitions for the specific file. Synchronized updating is a complicated procedure, and may be impractical for complex file structures.

Delayed and synchronized updating can be used where duplicate file copies are maintained permanently at different locations. It can also be used when a file copy is temporarily transmitted to another processor and is updated there as well as at the original location. This mode can also be used to allow updates to be performed in more than one processor simultaneously. Each processor would have assigned one or more record insertion areas, and each would converse with all other processors before activating any job requiring a current file version, since this would trigger application of delayed updates. Finally, synchronizing can be used, without delayed update, to allow simultaneous updating of multiple copies of a file, all of which are kept constantly current. Each such update requires a check throughout the network to see if synchronization is required, and a delay in further access until synchronization is accomplished, if needed. Update application can then be performed.

The ability to concurrently update one or multiple copies of the same file has many advantages. It, at least theoretically, allows great flexibility in assigning file locations, duplicating files, etcetera. The interference associated with concurrent updates is reduced by confining it to the period of time necessary to actually apply updates, rather than extending it over the time of execution of all jobs involved. Also, if updates are applied only on successful job completion, no rollback is ever required, since the main file is not changed.

This method also has several disadvantages. One is that file accesses are increased, since both a pseudo and actual update access are required for every successful update. Additional overhead is caused by the necessity to search the user's change areas on each new access, to determine whether he should be given his private copy or the common copy of the requested record. The use of delayed update also means that many users accessing the file will see a slightly out-of-date version of the

file data. Whether or not this is significant depends on the application involved. Finally, the synchronization required to allow multiple concurrent delayed update users is an extremely complicated piece of software, particularly for complex file types.

The type of application for which this mode seems suited is transaction processing, where there are often numerous requests for a few accesses to a file, many of which do not involve updating. Using the methods described, rapid, economical, and reliable response can be provided by multiple copies of the file, each of which can be updated concurrently. Retrievals from each copy are delayed only while updates are being applied, but the possibility that a subsequent retrieval will encounter different values has to be tolerated. Multiple copies can be provided at each of several processors, or the number of copies at each site can be adjusted to the volume of transactions expected there.

12.3.7.6 Duplicated Files

Unless controlled, simultaneous access by multiple programs to duplicated copies of the same file can cause differences between the copies. Depending on file content and use, it may be possible to minimize these problems by applying the following restrictions:

- Confine updating of all copies of a file to only one location at a time, because
 - Only specified personnel have the authority or knowledge required to change the file and they must control and coordinate updates to all copies
 - Certification and coordination of changes makes updating at one place at a time desirable
 - Volume of changes is so low in relation to retrievals that transmission of changes to a single location, and their posting there, can be tolerated.
- Allow retrievals from a version even though the most recent updates have not been applied.

12.3.8 DDB Reliability and Integrity

The DDB offers reliability since the DDB concept permits the use of at least a residual part of the data base, even though some parts have been lost or are inaccessible. The question of DDB reliability and/or integrity and which of the previously discussed DDB configurations is the most advantageous from the point of view of reliability and integrity has not been fully ascertained. Farber [CC] suggests that conventional reliability analysis is not adequate for a system which degrades to partial capacity when one or more of its components fail.

File integrity protection ensures that the data content of files, and file structures, are in no way altered or destroyed inadvertently. System integrity protection similarly protects job data and the operating system. Operating systems within network-resident processors must provide extensive integrity protection.

If an application updates files in more than one processor within a network, many integrity problems arise. File integrity protection, whether local or remote, generally takes one of two forms. It may involve journaling file changes, so that the file can be restored to a prior state if needed. Or, it may involve delayed updating, in which changes are posted only after known to be correct.

Since file protection involves overhead costs, it is generally optional. Files whose content is not significant, or which can easily be re-created, can be classified as unprotected. Damage to these files must be manually repaired.

System integrity is involved whenever a job in execution in one processor makes an access request to another location. Because system integrity is concerned with job/system status, it is responsible for keeping track of items such as outstanding remote requests. If any processor or communications link failure causes loss of contact between the processors involved, action must be taken to re-establish contact and re-synchronize all processes.

Logic is therefore required in all processors to handle remote access requests received just prior to a failure of the receiver, sender, and/or intervening

communications link. Generally, the method adopted is to ignore received requests assuming that the requesting processor will take all necessary action.

The logic to keep track of outstanding remote requests becomes more complex as the requests take longer to service. In the simplest case, a request for one record, a timer is usually sufficient to check for failure to respond. If an answer is not received in the expected time interval, a failure can be assumed in the other processor or in the communications link.

12.4 APPLICATIONS

12.4.1 Multiplexed Information and Computing Service (Multics)

Multics [AA] is the name given to a general-purpose computer timesharing system developed by Massachusetts Institute of Technology (MIT), Honeywell Information Systems (HIS), and Bell Telephone Laboratories. Multics permits the sharing of data within the framework of the system. This means that one may:

- Link with any other user's programs and data
- Move one's base of operation into another user's directory
- Access any data stored in the system.

The Multics system [X] also uses the directory concept. It considers the directory as a segment containing a list of the names of other segments. Each user has a directory for his own segment. Directories may be named, and a user's directory may contain the names not only of his segments, but also of additional directories he created. When a directory name is found in a directory, the directory containing the name is said to be inferior; the naming directory is superior. All the directories form a hierarchy, or tree (see Figure 12-11). The directory at the base of the tree which is the most superior is called the root directory.

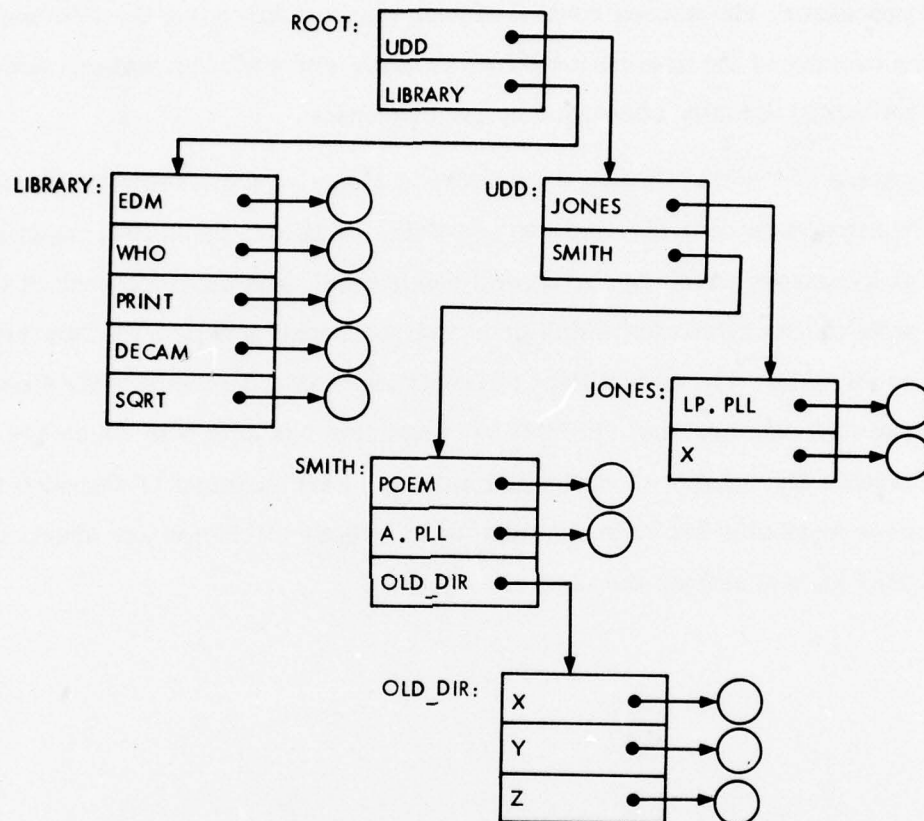


Figure 12-11. Typical MULTICS Directory Hierarchy
(Directories are Rectangles; Segments are Circles)

Access to each segment can be controlled by the owner of the segment, so that different users may have different access modes for the same segment. The owner also specifies those who may initiate and/or change the access mode.

In Multics, segmentation permits direct accessing and sharing of on-line data by a processor, and access control of such data. Addressing the information directly makes copying of the data no longer necessary, since all instructions and data in the system can be directly addressed by the processor.

Since all on-line data in the system is directly addressable by any operation, it is necessary to control access to this data. Multics, therefore, separates all data in a memory which can be directly addressed, and provides each of these data sets with access attributes which give each user instructions as to how he may reference the data. The capabilities of direct address and access control point the way to controlled data sharing. In Multics, segments are data sets which are directly addressable and subject to controlled access. Each segment is provided with a set of access attributes for each segment user. These attributes are checked before a user may have access to the data.

12.4.2 Advanced Research Projects Agency Network (ARPANET)

ARPANET was begun in the mid-1960's to explore the feasibility of remote access and sharing of computer resources. Since then the network has grown to span a large part of the globe from Hawaii to Europe with about 60 connection nodes and over 70 computers. ARPANET's main objective is to permit communications between, and processing of, data, programs, and services on diverse computers from any place in the network. The network is a full-duplex high-speed packet-switched data transmission network. Minicomputers at each node of the network, called Interface Message Processors (IMPs), are interconnected by 50-kbps leased lines and satellite links. The IMPs serve as controllers, message routers and statistical gathering devices.

In the ARPANET, up to four host computers, or hosts, may be connected to each node. When a message is sent, the IMP to which the host is connected breaks the message into discrete variable length transfer units, called "packets", of not more than a thousand bits. The packet-switching system, of which the sending IMP is a part, uses the destination address at the beginning of the message, to guide the packet through the network to its destination IMP. The IMP reassembles the packets into the original message and sends it to the destination host. Upon arrival, the system notifies the sender of its receipt.

ARPANET, in addition to the nodes with IMPs, has nodes called Terminal Interface Processors (TIPs). The TIP does not perform services, but provides for direct connection to terminal hardware, e.g., line printer, display, etcetera. Interactive terminal users attached to a TIP must obtain all their services (i.e., processing and storage) from ARPANET hosts. See Figure 12-12.

Each network computer uses a Network Control Program (NCP) which establishes and terminates connections between hosts, monitors program interchange, and controls other functions for user programs. Host computers in the ARPANET range from the PDP-11 through ILLIAC IV. See Figure 12-13.

AD-A041 459

COMPUTER SCIENCES CORP FALLS CHURCH VA
A COMPREHENSIVE DATA BASE ACCESS METHODOLOGIES DESIGN GUIDE.(U)

F/G 9/2

UNCLASSIFIED

SEP 76 J EMORY, S GREEN, R GRIMES, O HASLOP DCA100-75-C-0029

CSC-R493700019-2-2

CCTC-TM-123-76

NL

4 OF 4

AD
A041459



END

DATE
FILMED
7 - 77

12.4.3 Prototype World-Wide Military Command and Control System (WWMCCS) Intercomputer Network (PWIN)

The PWIN is being developed by the Command and Control Technical Center (CCTC) of the Defense Communication Agency to provide a test bed computer network for evaluating the utility of an intercomputer network for command and control activities. Initially, the PWIN will consist of six WWMCCS sites:

- CCTC Reston
- CCTC Pentagon
- CINCLANT (Norfolk, Va.)
- RED COM (Tampa, Fla.)
- MAC (St. Louis, Ill.)
- ANMCC (Ft. Ritchie, Md.)

Eventually, the system may interface with up to 35 WWMCCS computers located at military installations throughout the world.

Special host computer software will provide user access to the various computer network hosts. The technology employed in this initial development is ARPA based. A modified ARPA IMP will be utilized as the network's communications processor. The IMP is a modified Honeywell series 16 processor which provides packet switching communications technology.

Packet switching employs fast links and short data handling units (typically 1000 bits/packet) to cost-effectively accommodate man-computer, computer-computer, and computer-machine data traffic. Switching of packets provides superior performance at low cost for classes of traffic which involve many short messages requiring extremely rapid end-to-end delivery to many different users. It is a state-of-the-art refinement of classical store-and-forward switching techniques. Classical store-and-forward systems employ a variable length transfer unit up to that of the longest permissible message while recent store-and-forward switching systems employ a transfer length unit (packet) more tailored to the expected message length. The packet switches maintain the designed network speed of delivery through the use of traffic acceptance load control.

Each packet is individually routed and the IMP network employs adaptive routing. Adaptive routing allows the communications network to route around a failed communications processor and routes packets based upon algorithms that select the fastest delivery path. Note, because of adaptive routing at the individual packet level, packets may arrive out of sequence at the destination and, therefore, must be reassembled into the original message sequence.

Several enhancements to the original ARPA Network capabilities are incorporated in PWIN, such as additional levels of message packet precedence, increased survivability, and special interface functions:

The PWIN software includes:

- Network Control Program (NCP)
- TELNET
- Teleconferencing
- Network Command Language (NCL)
- File Transfer Protocol (FTP)
- Workload Sharing (WLS) Program.

The basic networking utility is provided by the NCP software. NCP contains the User-to-Network, Host-to-Network and Host-to-Host protocols. Through the use of these protocols, NCP can establish a connection from a program operating in one host to a program operating in any other host in the network.

TELNET provides further user access to the network. TELNET allows any user of an interactive terminal device to request connection to any direct access program in a distant host.

Teleconferencing provides a conference capability to terminal users, and, in conjunction with TELNET, serves both local and remote users. The structure of the conference consists of a chairman and many participants. The chairman controls the proceedings through various features, accessible only through him. One such feature is the capability of any participant to display his terminal activity to other

participants. To do this, the participant would request the floor, and the chairman would respond to the request. Basic features, available to all participants are: message entry, the ability to send messages to other participants; connection to other direct access programs, such as Timesharing, or a user program; reviewing previously entered conference messages; receiving printed copies of the conference minutes; and other conference related features.

The NCL allows users to integrate a sequence of operations on PWIN to form a cohesive processing unit called an NCL Job in much the same way as the operating system on a PWIN host (GCOS) provides for combining of separate user computing activities into a single unit called a Job. NCL statements provide users on any PWIN host to:

- Transfer a data file from one PWIN host to another
- Transfer a job from one to any other Host of the Network for execution with output resulting from the execution directed to any host
- Initiate workload sharing activity between hosts.

Furthermore, NCL provides PWIN users with control over their respective NCL Job executions through the use of a set of control statements which provide:

- NCL Job status information
- Cancellation directives
- Conditional control over execution of individual statements comprising an NCL job.

Finally, the NCL provides the user with a variety of recovery options which can be invoked to minimize the overhead associated with recovery from unscheduled system failures.

WLS provides the network the ability to load level. WLS is a mode of operation available to each host site. Each host's operator can specify that his site is in WLS mode or not. When in WLS mode, jobs will be scanned for shareability (shareability

is specified within the job by the job originator). If shareable, alternate network hosts are polled to determine if they will accept the job. The WLS protocols include sufficient exchange of data to define the resource requirements of the job so that each host can validly accept or reject a job. If an alternate host is found, the job is sent to the first alternate site that responds with an "I will accept" message. Prior to sending the job, the local host operator is given the option of terminating the job or spawning* the job locally. If no alternate site is found, the local operator may exercise a local option.

12.4.4 Distributed Computer System (DCS)

The DCS is an experimental computer network developed by the University of California. It consists of a series of minicomputers connected to each other by a unidirectional digital communication ring using ring interface (RI) hardware. The ring interfaces are of three types: one which supports a computer, another which permits the direct attachment of a terminal to the ring, and a third which allows the construction of additional rings or a network of rings.

There is no designated processor for network control. Control is distributed throughout the processors in the network. One of the design objectives of the system was to develop a reliable, fail-soft network, which could tolerate the failure of any component without seriously interrupting the rest of the network. Since control functions are distributed in time and space, the control will move from processor to processor, depending upon the load of the processor.

Messages are addressed to process names, but routed by hardware. As previously stated, each processor is connected to the communication ring by an RI. The RI compares the destination address of each message which passes in the ring with the process names stored in its memory. When a match occurs, the message is passed to the process stored in the processor and also sent on along the ring.

Should a user or process require additional resources, it can send out a Request for Quotation (RFQ). Each processor has a Resource Allocator which allocates its resources. When a RFQ is received by a Resource Allocator (all Resource Allocators recognize a general name), the Resource Allocator "bids" on providing the

*A Honeywell term for initiating a separate and independent execution.

requested resource. The requesting processor evaluates all received bids, selects the best bid according to an algorithm, and sends a "contract" to the selected processor. The latter may or may not accept the contract, depending upon its workload. The requesting processor must wait for acceptance of the contract before obtaining the desired service.

12.4.5 ALOHA Network

The ALOHA Network (ALOHANET) is an experimental UHF radio packet-switched computer communication network developed by the University of Hawaii. It is the first system which successfully has applied the packet-switching concept to online access of a computer via radio. The ALOHANET uses a new form of burst random access communication technique, and a form of multiplexing using two 24-kbps channels for all remote units to interface with a central IBM 360/65 processor. Since multiplexing is done automatically, the system can handle over 500 active user terminals.

The notion that many widely dispersed users can send data packets over a single high speed communication channel is supported by the fact that users will send packets infrequently and each packet can be transmitted in an interval of time which is considerably less than the average time between packets. Each packet has a header attached which contains the destination address, control information, etcetera. This packet is sent by each user to the central location over the same channel in an unsynchronized fashion. If errors occur, only error-free packets are accepted by the central station, the transmitting station automatically retransmits the packet until an acknowledgement is received, or until a certain number of unsuccessful attempts have been made.

The ALOHANET is linked to the ARPANET using a single voice channel on the INTELSAT IV satellite. Such a channel, used in a packet-switched mode, can be used for receiving, transmitting, or "bilateral broadcasting".

12.4.6 TELEX Switching System

The TELEX Switching System is a commercial solid-state, stored-program computer-controlled telephone exchange network sold by Astrodata. Its capabilities support a range from private systems to large switching systems composed of 2 to 16 control units, in a distributed load-sharing configuration. The network is arranged to allow any one of the terminators to communicate with any other terminator without restriction. Since there are multiple redundant processors, the system will operate even though failure may occur in some part of it. Reliability is ensured through dual redundancy, continuous automatic performance monitoring, built-in diagnostic testing, etcetera.

System hardware provides the electrical interface between software and the TELEX network and translates electrical pulses received from the TELEX System into a form suitable for software processing. The software performs the routing, makes decisions, and performs the character transfer functions. The software is stored in a semi-permanent magnetic core memory (8-65K words) in each processor. Each word is 16 bits and contains reference information, transient information, or a single instruction.

When a call is initiated, a two-word data transfer is stored into the input area of the memory until previously arrived data is processed. As soon as this is accomplished, the software reads the first word which contains the address of the terminator which sensed the off-hook condition. The second word is then read, and a Call Processing Register (CPR) and Call Register (CR) are assigned. The contents of the two registers are set and updated. As input selection signals are received, these signals are processed by the CPR and CR. The software selects the subscriber or trunk terminator addressed and makes another CPR available. This CPR is linked to the previously designated CR permitting transfer from one terminator to another. Once the link is established, the software releases the two CPRs and enters into a "character coupling" mode which permits the actual transfer of the TELEX characters. Upon a disconnect from either party, the software will restore the registers, tables, etcetera to their previous state.

12.4.7 AUTODIN II

AUTODIN II, an updated version of the Defense Department's Automatic Digital Network, AUTODIN I, is a packet-switched intercomputer network. Each packet is synonymous with a message and consists of up to 2000 bits. The system will use third generation computers connected by high-speed telecommunications lines. The Network Control Program (NCP) resides within the network and not within a particular computer. This provides for enhanced reliability since failure of a processor or part of the network will not impair the functions of the rest of the network. The system, having 16 levels of precedence, ensures the transmission of high priority messages and assigns other messages lesser priorities. The network is designed so that other separate networks, such as TELENET and ARPANET, may be integrated into AUTODIN II if so desired.

CHAPTER 13 - ABBREVIATIONS AND ACRONYMS

AP - Application Programmer
ARPANET - Advanced Research Projects Agency Network
ASCII - American Standard Code for Information Interchange
AUTODIN - Automatic Digital Network
BCD - Binary Coded Decimal
BPI - Bytes Per Inch
CCTC - Command and Control Technical Center
CODASYL - Conference on Data Systems Languages
CPI - Characters Per Inch
CPR - Call Processing Register
CPU - Central Processing Unit
CR - Call Register
CRT - Cathode Ray Tube
CTMC - Communications Terminal Module Controller
DASD - Direct Access Storage Device
DBA - Data Base Administrator
DBL - Data Base Language
DBMS - Data Base Management System
DBTG - Data Base Task Group
DCS - Distributed Computer System
DDB - Distributed Data Base
DDL - Data Description Language
DML - Data Manipulation Language
DMS - Data Management System
DSL - Data Sublanguage
DTP - Data Transfer Protocol
EBCD - Extended Binary Coded Decimal
EBCDIC - Extended Binary Coded Decimal Interchange Code

FIFO - First In First Out
FMS - File Management Supervisor
FTP - File Transfer Protocol
GCOS - General Comprehensive Operating Supervisor
I-D-S - Integrated Data Store
IMP - Interface Message Processor
IMS - Information Management System
INFONET - Information Network
I/O - Input/Output
IOCS - Input/Output Control System
i p s - inches per second
ISAM - Indexed Sequential Access Method
ISP - Indexed Sequential Processor
JCL - Job Control Language
LIFO - Last In First Out
MSS - Mass Storage System
MULTICS - Multiplexed Information and Computing Service
NAP - Network Accounting Program
NCL - Network Command Language
NCP - Network Control Program
OCR - Optical Character Recognition
PDL - Physical Description Language
PWIN - Prototype WWMCCS Intercomputer Network
RCC - Remote Computer Concentrator
RFQ - Request For Quotation
RI - Ring Interface
RJE - Remote Job Entry
RSN - Relative Sequence Number
RTS - Remote Teleprocessing Service
SA - Systems Analyst
SCP - System Control Processor

TIP - Terminal Interface Processor
TBM - Terabit Memory
TDS - Transaction Driven System
UHF - Ultra-High Frequency
VIP - Visual Information Processor
VSAM - Virtual Storage Access Method
WWDMS - World Wide Data Management System
WLS - Work Load Sharing
WWMCCS - World Wide Military Command and Control System
1NF - First Normal Form
2NF - Second Normal Form

CHAPTER 14 - GLOSSARY

Aggregate	Collection of organized data elements
Attribute	Informational analog of a real-world property
Bank	Totality of extents
Binary Search	A technique for finding an item in a file by dividing the file into successive halves until the item is found or the search fails
Bit Map	Association of a given key value with a series of bits indicating whether a key value is within a record or not
Bit Vector	See Bit Map; indicates whether or not a key value is within a set (or block) of records
Block	<ol style="list-style-type: none">1) A physical record read or written on a storage medium as the result of a single I/O command2) A group of characters on a magnetic tape
Bucket	Small storage section
Catalog	An ordered compilation of data items with item descriptions and physical addresses of records
Chains	See Linked Lists
Core Storage	Usually main memory of a CPU
"Current" Element	First element in a structure where an operation starts
Cylinder	Series of tracks on a stack of discs
Data	Discrete units or the total information stored within a system
Data Bank	See Bank
Data Base, Logical	A set of one or more logical data files
Data Element	A set of one or more logical data items
Data, Organized	See Organized Data
Data, Physical	See Data
Deque	A serial structure referenced from the "leftmost" or "rightmost" element
Dictionary	A data structure containing definitions and descriptions of data
Direct Address	An address obtained directly from the primary key of a record

Directory	Describes the relationships between the data base, its files, data sets, records & fields (distributed or non-distributed)
Domain	A set of all values for a particular attribute associated with a given entity-set
Enterprise	A global conception of an entire data system
Entity	Object or relation about which information is desired
Entity Set	A collection of similar entities
Extent	A contiguous region on an external storage medium for the storage of a physical record
Field	The smallest element of an organized data structure used for a particular category of data
Files, Logical Data	A set of one or more logical data records
Files, Physical	All extents within a file
Frame	A row of bits across a magnetic tape
Hashing	An access method which uses an algorithm on the access key to a record to determine the address of the record
Hosts	The interconnected computers of a computer network
Index	A table containing keys and pointers
Information	Processed data
Insertion	To add a new record to a file
Item, Logical Data	The smallest named component of a logical structure
Linked Lists	Data elements connected by pointers
Logical Data Base	See Data Base, Logical
Logical Data Files	See Files, Logical Data
Logical Data Item	See Item, Logical Data
Logical Data Record	See Record, Logical Data
Logical Structure	A model of the real-life relationships of data
Machine Address	Actual location of data on its storage device
Model	A view of the organized data in its totality
Network	A set of interconnected computer systems
Network Structure	Structure in which any node may be linked to any other node; synonymous with plex structures

Node	The representation of a state or event by means of a point on a diagram
Organized Data	Information expressed through relationships e.g., schemas, models
Packet	A predetermined size segment of a message
Password	Identifier used as a part of a validating sequence
Peripherals	Auxiliary equipment under the control of the central computer
Physical Data	See Data
Physical Files	See Files, Physical
Physical Record	See Block
Physical Storage	The actual placement of data on a storage device
Plex Structure	See network structure
Pointer	A field within a record which indicates where another record is located
Probe	Each access to an index
Protocol	A set of rules or procedures which facilitate communication throughout a network
Queue	A serial structure referenced from the "front" or from the "rear" representing a FIFO list
Real-World Information	Information not constrained by the limits of computer technology
Record, Logical Data	A set of one or more logical data elements
Record, Physical	See Block
Relational Structure	A structure consisting of sets of related values
Relative Address	Location of a record in relation to the beginning of a block or page of records, which may or may not be stored contiguously
Relative Sequence Number	Denotes position of a logical data element in a list
Schema	A description of organized information
Secondary Storage	All storage devices other than the computer main storage
Sector	Smallest addressable unit on a drum or disc
Serial Structure	Set of logical data elements with linear (one-dimensional) relationships

Stack	A serial structure referenced from the "top" representing a LIFO list
Submodel	A programmer's or system analyst's view of the enterprise
Subschema	Subset of the organized data described within a schema
Track	<ol style="list-style-type: none"> 1) Parallel rows of bits along a magnetic tape 2) Concentric circles of bits on a disc 3) Parallel circles of bits around a drum
Transaction Journal	A record of every transaction entering a system
Tree Structure	A series of hierarchically arranged nodes
Trie Structure	A tree structure where the nth node contains the nth character or digit representation of an incoming key

REFERENCES

This bibliography is constrained to include only those references which are specifically referenced in the text. Many of the references contain extensive bibliographies of their own - with considerable overlap. Several (e.g., Date[F], Knuth[H], and Martin[G]) contain annotated bibliographies which are of significant help in determining a next best source.

- (A) E. F. Codd, "Normalized Data Base Structure: A Brief Tutorial", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, obtainable from ACM HQ, New York, New York.

This paper is a tutorial in which the relational model is explained. Advantages with respect to the hierarchical, network, and cross-referencing structures are discussed. Finally, the concept of normalization is explained and illustrated. The paper and reference (B) served as principal sources for Chapter 9.

- (B) E. F. Codd, "A Data Base Sublanguage Founded on the Relational Calculus", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, obtainable from ACM HQ, New York, New York.

This paper describes a high level relational calculus-based data sublanguage. Concepts and principles are stressed. Arguments are presented for the superiority of the calculus-based type of data base sub-language over the algebraic, and for the algebraic over the low-level procedural.

- (C) E. F. Codd, "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposia and Data Base Systems, Prentice-Hall, May 1971.

This paper defines a relational algebra and a relational calculus, and proves that the algebra has at least the retrieval power of the calculus. An algorithm is presented for converting an arbitrary calculus expression into an equivalent algebraic expression.

- (D) CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems", May 1971, CODASYL Systems Committee Report, obtainable from ACM HQ, New York, New York.

This report contains a state-of-the-art discussion of data management capabilities, technical problems facing designers, generalized data management systems, and definitions. Finally, a feature comparison is given for IBM's GIS, IMS, and NIPS/FFS; Honeywell's I-D-S; Informatics' MARK IV; Auerbach and Western Electric's SC-1; SDC's TDMS; and RCA's UL/1.

- (E) R. W. Engles, "A Tutorial on Data-Base Organization", Annual Review in Automatic Programming, Volume 7, Part 1, 1972, Pergamon Press, 1972.

This paper is a tutorial on data base organization. The main issues are data independence, security, integrity, search, and the integrated data base. Data Management history, trends, and terminology are also included.

- (F) C. J. Date, "An Introduction to Data Base Systems", The Systems Programming Series, Addison-Wesley Publishing Company, 1975.

This volume is in the nature of an introductory textbook on data bases and data base systems. It covers data base system architecture, the relational approach, the hierarchical approach, the network approach, security, and integrity. This text was used as the principal source for Chapter 8 of Part III.

- (G) James Martin, "Computer Data Base Organization", Series in Automatic Computation, Prentice-Hall Inc., 1975.

This volume is intended to be used as a basic text for a course on data base technology. It is the principal source of information for Part II, Chapters VI and VII. It acquaints the reader with the many alternatives possible in data organization and the tradeoffs between them.

- (H) Donald E. Knuth, "The Art of Computer Programming", Volumes 1 through 3, Addison-Wesley Publishing Company, 1973.

This is a series of books designed to train the reader in the various skills which go into a programmer's craft. These books are used as texts for college courses in computer and information services, but can also be used for self study.

Volume 1 covers fundamental algorithms.

Volume 2 covers seminumerical algorithms.

Volume 3 covers sorting and searching.

This series contains an exhaustive treatment of topics covered.

- (I) "COBOL Compiler, Series 600/6000 Software", Order Number BS08, Honeywell Information Systems, May 1971.

This is a technical manual containing a complete description of the Common Business Oriented Language (COBOL) compiler implemented specifically for Series 600 and 6000 Honeywell Information Systems.

- (J) E. F. Codd, "Recent Investigations in Relational Data Base Systems", Information Processing 74, North Holland Publishing Company, 1974.

This paper discusses the objectives and characteristics of the relational approach to management of large, formatted, and integrated data bases. It also considers advances in the following topics:

- Normalization of the relational model
- Data base sublanguages for programmers and nonprogrammers
- Superimposition of submodels in the relational model
- Data exchange policies in a network of mutually remote data bases.

- (K) Sham Navathe, "Logical Normal Forms For Data Translation", Data Translation Working Paper 802, Contract Number DCA 100-75-C-0019, Defense Communications Agency, Department of Defense, Washington, D.C., January 1974.

This working paper discusses the effects of normalization in data translation. It also reviews normal forms as defined by Codd and operations on Codd's normal forms.

- (L) Shamkant B. Navathe, "Investigations into the Application of the Relational Model to Data Translation", Data Translation Paper 808, Contract Number DCA 100-75-C-0019, Joint Technical Support Activity, Defense Communications Agency, Department of Defense, Reston, Virginia, December 1974.

This working paper discusses normal forms in data translation and describes a model for data translation using normal forms.

- (M) Bachman, Charles, "Integrated Data Store", DPMA Quarterly, January 1975.

This paper is the first major description of the Integrated Data Store (I-D-S) data base management system. The basic principles of the design and implementation of I-D-S are presented in a brief summary.

- (N) Doug Barteck, "Systems Surveillance and Monitoring", April 1975, Proceedings of the Computer and Privacy Symposium, Honeywell Information Systems, Brighton, Massachusetts.

This paper details the methods used to detect the presence of penetration attempts. The basic methods are surveillance and monitoring techniques. Two types of monitors are presented, active and passive. The methods for collection of data and resultant countermeasures for each type are discussed. Also, general prevention concepts are listed.

- (O) "Integrated Data Store", Series 600/6000 Software, Order Number BR69 (Rev. 1) Honeywell Information Systems, December 1971.

This manual is the reference manual for I-D-S programming and systems analysis. The constructs required for manipulating I-D-S data structures are detailed.

- (P) Honeywell Information Systems, "World Wide Data Management System", DB98.

This manual describes the basic concepts of the World Wide Data Management System. It is intended to be used by data base administrators and system designers. Basic data structures and system interactions are described.

- (Q) Peter S. Browne and Dennis D. Steinauer, "A Model for Access Control".

This paper presents the fundamental problems of any security system implemented on a computerized system. A discussion of the requirements for a multiple-user information system are detailed. From these requirements, a model is developed. The purpose of this model is to enable future designers to obtain the proper concepts for an information processing system.

- (R) Arthur Evans, Jr., William Kantrowitz, Edwin Weiss, "A User Authentication Scheme not Requiring Secrecy in the Computer", Communications of the ACM, August 1974.

This paper investigates the method by which a user's password is compared against a table of stored passwords. A transformation algorithm is used in which the resultant password cannot be reversed into the original password. The main problem discussed is the determination of the algorithm to be used for the transformation.

- (S) L. Smith, "Architectures for Secure Computing Systems", Mitre Report AD-A009 221, April 1975.

This paper describes the basic issue of security and how five large scale systems compare. The architectural features of a system which relate to security are listed. The machines are then compared to these features to produce an effective security rating of architectures. The computer systems included in the study are Honeywell 6180, Burroughs B6700, DEC System KI-10, IBM 370 and the XDS Sigma 9. The machine judged most desirable is the Honeywell 6180.

- (T) C. Weisman, "Security Controls in the ADEPT-50 Time Sharing System", 1969 FJCC, AFIPS Conference Proceedings, Vol 35, pp. 119 through 133.

This paper presents the fundamental concepts used to develop the ADEPT-50 timesharing system. A mathematical model was developed before the implementation of any software. The model was designed around security controls required for adequate information protection. The software was then designed around this model to operate on an IBM 360 model 50.

- (U) R.D. Lackey, "Penetration of Computer Systems, An Overview", Honeywell Computer Journal, Vol 8, No. 2.

This paper presents methods for attacking and penetrating computer systems for unauthorized purposes. These techniques are categorized to produce a set of generalized concepts applicable to all computer systems. Also, preventative methods are presented to counter such penetration attempts.

- (V) "Integrated Data Store", Series 600/6000 Software, Order Number BR69 (Rev. 1) Honeywell Information Systems, December 1971.

This manual is the reference manual for I-D-S programming and systems analysis. The constructs required for manipulating I-D-S data structures are detailed.

- (W) Edwin J. McCauley and Peter A. Alsberg, "Research in Network Data Management and Resources Sharing - Scenario Report", Center for Advanced Computation, University of Illinois at Urbana - Champaign, for JTSA, CAC Number 159, JTSA number 5506, May 19, 1975.

This is an internal working paper which presents a limited view of some major research questions together with a restricted discussion of distributed data design alternatives.

- (X) "The Multics Virtual Memory", Multics Technical Papers, Honeywell Information Systems, Number AG95, June 1972.

This reference consists of three technical papers which describe the theory and

practice behind the implementation of Multics virtual memory scheme. The first paper discusses the concept of virtual memory. The second paper extends the discussion to include the subject of protection. The last deals with hardware under the optional Multics modifications to Series 6000 processors.

(Y) Reserved

(Z) Grayce M. Booth, "The Use of Distributed Data Bases in Information Networks", Proceedings of the First International Conference on Computer Communication: Impacts and Implications, October 24-26, 1972.

This paper presents the following:

- An overview of theories relating to distribution of a data base within a computer network
- Alternate methods for distributing data
- Manual and software methods for matching data files with data bases
- Accessing of components of a distributed data base
- Problems associated with security and system integrity as they relate to distributed data

It presents theoretical work in a straight forward easy-to-understand fashion.

(AA) "Programmers Manual, Vol 1 - Introduction", Honeywell Information Systems and MIT, Number AG90, Rev. 1, May 1973.

The Multics Programmer's Manual (MPM) is the primary reference manual for user and subsystem programming on the Multics system. It is divided into five volumes as follows:

<u>Volume No.</u>	<u>Title</u>	<u>Document No.</u>
I	Introduction	AG 90
II	Reference Guide	AG 91
III	Commands and Active Functions	AG 92

<u>Volume No.</u>	<u>Title</u>	<u>Document No.</u>
IV	Subroutines	AG 93
V	Subsystem Writer's Guide	AK92

This reference is to the Introduction, Volume I.

- (BB) "Distributed Data Base Concepts and Requirements", Computer Sciences Corporation, for JTSA, May 1, 1973.

This is a report which presents concepts and requirements concerning the implementation and use of distributed data bases with emphasis on implementation alternatives and potential problems.

- (CC) David J. Farber, "Distributed Data Bases, An Exploration", Department of Information and Computer Science, University of California, Irvine, Ca.

This paper is devoted to an exploration into problems affecting the management, cost, reliability, security, and transferability of distributed data systems. The feasibility of distributing data is also discussed.

APPENDIX A - MATHEMATICAL MODEL

The basic symbols and equations used are obtained from the work done by Browne and Steinauer [Q] to maintain uniformity and continuity with existing literature.

The definition of the items which are to be protected are the objects. The objects (O) which are of most interest in developing the model are processes (p), terminals (t), data (d), jobs (j)*, and users (u). That is :

$$O = \{p, t, d, j, u\}$$

The distinction between objects is important during the development of the model to indicate the current level of security required.

The level (l) describes a hierarchical structure indicating relative degrees of access. By definition, the highest level has access to all lower levels. The standards imposed within the military are Unclassified, Confidential, Secret, and Top Secret. For an object "i"

$$l_i = \{l_0 \leq l_1 \leq l_2 \leq \dots l_n\}$$

or for specific military applications

$$l_i = \{\text{Unclassified} \leq \text{Confidential} \leq \text{Secret} \leq \text{Top Secret}\} .$$

The designation of level does not provide enough control since all individuals with a Top Secret clearance should not view all documents. Therefore, categories (k) (compartments) are designed to restrict specific groups of users. For example, a category may be the "Army only", or "President only", etc. Thus, each group is independent and has no higher rank than any other. Therefore, for object "i"

$$k_i = \{k_0, k_1, k_2, \dots k_n\}$$

A protection group can be determined from the combination of level and category. Thus, a protection group (g) is defined as the ordered pair

$$g = [l_i, k_i] .$$

* A job is the overall activity which may contain or access many processes. The job's attributes will change dynamically throughout time.

For the specific pair [Secret, President only], the access is limited to the President if, and only if, he has either Secret or Top Secret clearance.

Two criteria must be met to ensure that the object can be accessed by the requestor. The first criteria states to which protection groups the object belongs. This group is considered the classification (C) of the object. For an object "i",

$$C_i = \left[l_i^c, k_i^c \right].$$

To determine who has access to an object, the subject (requesting object) must possess a similar set of protection groups. This group is considered the authority (A) of the subject. Thus, for subject "i",

$$A_i = \left[l_i^a, k_i^a \right].$$

The formal definition of accessibility between classification and authority will be presented later, but intuitively the example of the [Secret, President] indicates the comparison between authority and classification.

There is a third attribute (other than level and category) which controls the second criteria for access. The third attribute states that those subjects with a high enough level and falling within the correct category must also have a "need-to-know" to access the object. The access list (L) (franchise) specifies all subjects which may have access to the object and under what circumstances. This access list is a variable number of sets of capabilities (a,r). Therefore, for object "i", its access list is

$$L_i = \left[(a_1, r_1), (a_2, r_2), \dots (a_n, r_n) \right].$$

The components of the capability are the access clique (a) and the privilege list (r). The access clique is the set of subjects (s) which are allowed access to the object. Normally this list is composed of userids utilized for log-on control. Therefore, for object "i", its access clique is

$$a_i = \{ s_1, s_2, s_3, \dots s_n \}.$$

The privilege list is composed of a list of pairs indicating the permission (r) the subject has over the object and an indicator (q) designating if that permission can be passed on to the subject. The permissions typically used are read (r_r), write (r_w), execute (r_e), modify (r_m), create (r_c), and append (r_a). Thus, for object "i", the privilege list is

$$r_i = \left[(r_r, q_r), (r_w, q_w), (r_e, q_e) \dots (r_a, q_a) \right].$$

Therefore, the access list indicates who can access the object and under what conditions. Using the previous example, the President's name must appear on the document's access list with at least read permission. The use of permissions to physical objects is not mandatory but the internal processing of a computer requires such a mechanism. In summary, each object "i" has three attributes: authority, classification, and access list.

The access criteria can now be explicitly stated with these basic definitions. First, the authority of subject "i" must be equal to or greater than the classification of object "j" and, second, "i" must belong to the access list of "j" with proper permissions. Thus,

$$A_i \geq C_j$$

$$\text{and } (s_i, r_i) \in L_j$$

The determination of $A_i \geq C_j$ is made by comparing the individual components, level and category. The level of the subject must be greater than or equal to the level of the object and all categories of the object must be contained within the categories of the subject. Thus,

$$l_i^a \geq l_j^c$$

$$\text{and } k_j^c \equiv k_i^a \cap k_j^c$$

From this access criteria, the mechanisms through which activity must proceed can be defined. Each step defines checks which must be passed to allow further processing of the activity. The initial connection of a user to a system can be made if

the authority of the user is greater than or equal to the classification of the terminal. Also, the user must be within the access list of the terminal. Hence,

$$A_u \geq C_t$$

and $(s_u, r_u) \in L_t$

i.e., a Secret user cannot operate a Top Secret terminal.

The initial job has an authority of the user and no classification. The classification will change to reflect the highest classification accessed. The criteria for a job to access a process is similar to the general criteria with an additional restriction in the access list. The user must be contained within the access clique and at least the execute permission must be granted. Thus,

$$A_j \geq C_p$$

and $(s_u, r_u) \in L_p$.

As previously mentioned, the job takes on the highest classification of any process. This method of determining classification is called the "high water mark", the highest classification reached. Thus,

$$C_j \leftarrow \max (C_j, C_p).$$

A job at a present classification of Secret will obtain a classification of Top Secret during and after executing a process of Top Secret classification. Authority can be either assigned to a process or may be granted from the subject using the process. Thus, the authority for a process, A_p , is the maximum of the authority of the job, A_j , and the authority passed from the job, A_s . Therefore,

$$A_p \leftarrow \max (A_j, A_s)$$

One process may call another during the life span of a job. Careful control of security attributes must be maintained to ensure that correct access controls are imposed.

For process p_1 to call process p_2 , the authority of p_1 must be greater than or equal to classification of p_2 . Also, either the user must be included in the access list of p_2 or else process p_1 must be in the access list of p_2 . Therefore, for process p_1 to access process p_2^*

$$A_{p1} \geq C_{p2}$$

$$(s_u, r_u) \in L_{p2} \cup p1 \in L_{p2} .$$

Objects can be accessed through another process p_2 by allowing either the user or process p_1 to be in the access list of process p_2 . The classification of process p_1 must reflect that of both processes p_1 and p_2 to maintain the high water mark concept. Also, the classification of the job must reflect such changes. Thus,

$$C_{p1} \leftarrow \max (C_{p1}, C_{p2})$$

$$C_j \leftarrow \max (C_{p1}, C_j)$$

The authority of process p_2 must be determined and may have been granted by process p_1 .

The access criteria for data, d , is identical to that described for a process. Many systems, especially virtual systems, consider data synonymous with a process. Thus, to briefly state the equations for data access

$$A_p \geq A_d$$

$$u \in L_d \cup p \in L_d .$$

The privilege lists must also be considered, ensuring that proper permissions are granted when accessing data. In addition, the classification of the process and the job must change to reflect the classification of the data accessed.

*The interprocess control mechanism is fundamental in the architecture of the multiple domain operating system described in Paragraph 11.5.2.

$$C_p \leftarrow \max (C_p, C_d)$$

$$C_j \leftarrow \max (C_p, C_j)$$

The classification of any new data which is created must reflect the high water mark.

Thus,

$$C_d \leftarrow \max (C_j, C_p)$$

The determination of the high water mark for data is a simple procedure consisting of a few equations based upon the subjects who have accessed it. In reality, the classification should reflect the actual contents of the data. A Secret process may create unclassified data but it must be classified Secret to preserve the high water mark. The only means currently available to correctly declassify such data is a manual declassification procedure. The content of data can not be considered for classification purposes until processes can be certified to ensure that the data produced cannot be incorrectly classified. This concept, contextual sensitivity, has not been developed far enough to be fully used and needs further investigation.

INDEX

Absolute Address 5-9, 7-1, 7-10
Access 4-12, 5-19, 12-26, 12-34, 12-35, 12-37, 12-38, 12-39, 12-42, 12-44, 12-48
Access Authority 11-2
Access Control(s) 11-2, 11-13, 11-14, 11-20, 12-44
Accessibility of Data 11-3
Access List(s) 11-8, 11-14, 11-15
Access Mode(s) 12-36, 12-44
Access Time 2-4, 2-7, 2-14, 2-15, 2-16, 2-17
Adaptive Routing 12-49
Additions 10-7
Addressing Technique 6-1
Aggregates 1-11, 1-15, 1-16, 1-18
Algebra Based Sublanguages 8-9, 8-12, 8-16
Algebraic Coding 7-5, 7-6
Algorithmic Techniques 3-1, 3-4, 4-7, 4-8, 11-17, 11-18
Allocation 1-20
ALOHA 12-51, 12-52
Analog Transmission 12-3, 12-4
Application Programmer(s) 1-1, 1-6, 1-15, 1-16, 6-1, 6-21, 8-1
ARPANET 12-45, 12-52, 12-54
Artifice 11-9
Attribute(s) 1-11, 1-13, 1-18, 5-17, 6-44, 8-3, 8-5, 8-6, 8-14, 9-9, 9-11, 11-15, 12-44
Audit Trail(s) 10-4, 12-33
Authority Code 11-22
Authorization 11-15
Authorized User 11-9, 11-16
AUTODIN 12-54

Bachman 6-6, 7-14
Backup(s) 10-2, 10-4, 10-5, 10-7, 12-26, 12-28
Balanced Tree 6-12
Band 2-7
Bank(s) 1-19, 1-21
Beginning-of-Data 5-4
Binary Search 5-5, 5-7, 5-10, 5-13, 5-19, 5-21, 7-2
Binary Tree(s) 6-12, 6-14
Bit Map(s) 5-19, 6-46, 6-51, 6-52, 11-15
Bit Vector(s) 5-19, 6-46, 6-49
Block(s) 1-19, 1-20, 2-6, 4-9, 4-10, 4-15, 5-9, 5-10, 5-13, 5-15, 5-17, 5-19, 5-21,
5-22, 6-3, 7-10, 10-9, 12-35
Block Control 4-9
Blocking 4-9, 5-21

Block Search 5-13
 Boolean Algebra 6-49, 8-16, 8-17, 12-33
 Broadcast Media 12-4
 Browsing 11-8, 11-9
 b-Tree 6-14
 B-Tree 6-17, 6-19
 Bucket 5-9, 7-7, 7-8, 7-9, 7-10, 7-12
 Buffers 2-2, 8-19, 12-3, 12-5, 12-7, 12-9

 Calculus Approach 8-18
 Calculus Based Sublanguages 8-9, 8-12
 Call Processing Register 12-53
 Call Register 12-53
 Capacity 2-15, 2-16, 2-17
 Catalog 1-6, 1-8, 1-21, 6-1, 6-44, 12-31
 Cathode Ray Tube 2-2, 12-2
 Central Processing Unit 2-2
 Centralized Network 12-19, 12-20
 Chain(s) 4-16, 5-9, 5-15, 6-6, 6-8, 6-28, 6-30, 6-33, 6-37, 6-38, 6-44, 6-50, 6-51, 6-52, 6-53, 7-6, 7-7, 7-9, 7-12, 7-14, 12-37, 12-38
 Chain Master 6-8
 ChainPrior 6-8
 Channel 2-2, 12-4, 12-5, 12-6, 12-7, 12-51
 Checkpoint 10-3
 Checksum(s) 12-9
 Circuit(s) 12-3, 12-4, 12-5
 Circuit Switching 12-4, 12-5, 12-7
 Clustering 7-2
 COBOL 1-2, 1-5, 1-15, 1-18, 4-12, 6-50
 Collection File(s) 10-9, 10-10
 Collision(s) 7-2, 7-3, 7-6
 Command and Control 12-48
 Communication(s) 1-2, 12-2, 12-4, 12-5, 12-11, 12-13, 12-16, 12-19, 12-20, 12-21, 12-24, 12-41, 12-42, 12-45, 12-51
 Communication Network 12-1, 12-2, 12-3, 12-4, 12-48, 12-49, 12-51
 Communication Security 11-18
 Communication Software 12-1
 Communication System(s) 12-1
 Completely Connected Network 12-19 12-20
 Complex Map 6-3
 Complex-Complex Map 6-6

Complex Relationship 6-21
Complex-Simple Map 6-6
Concatenated Key 5-8
Conditional Expressions 8-17
Consecutive Spill Method 7-8
Control Permission(s) 11-15
Control Word 5-16
Cost 2-16
Cycle 6-22
Cylinder(s) 2-9, 5-13, 5-15, 5-22

Data 1-11, 1-13, 1-14, 1-21
Data Access 11-1, 11-2, 11-15
Data Administrator (See Data Base Administrator)
Data Bank(s) 1-20
Data Base(s) 1-19, 6-1, 6-33, 6-44, 6-50, 7-6, 7-9, 8-1, 8-8, 9-1, 9-6, 10-1, 10-3,
10-4, 10-6, 12-1, 12-21, 12-25, 12-33, 12-35, 12-38, 12-41
Data Base Administrator 1-1, 1-5, 1-6, 1-13, 1-15, 1-16, 1-19, 6-1, 8-1, 10-6, 11-22
Data Base Language(s) 1-2, 1-8, 1-16
Data Base Management System(s) 1-1, 1-2, 6-22, 6-25, 6-26, 6-44, 6-50, 7-7, 7-11,
8-18, 8-19, 10-1, 10-2, 10-3, 10-5, 10-6, 11-21, 11-22
Data Base Segment(s) 12-25
Data Base Task Group 1-2, 11-21, 11-22
Data Cell(s) 2-11, 2-15, 2-16, 2-17
Data Definition Language 11-22
Data Description Language(s) 1-2, 1-5, 1-6, 1-16
Data Management System 1-1, 1-2, 1-6, 1-8, 1-20
Data Manipulation Language(s) 1-2, 1-5, 1-6
Data Set(s) 12-33
Data Sharing 12-44
Data Sublanguage 8-19, 8-20
Data Transfer Protocol 12-49
Deadlock(s) 10-8, 10-9
De-allocation 1-20
Decryption 11-18
Delayed Update 12-38, 12-39, 12-40, 12-41
Deletion(s) 3-4, 4-1, 4-2, 4-4, 5-1, 5-3, 5-16, 5-22, 6-17, 6-27, 6-30, 6-51, 6-52,
6-27, 7-10, 9-1, 9-9, 10-7
Delimiter(s) 4-2, 4-4, 5-2, 5-5
Density 2-5, 2-15
Dequeue(s) 3-1, 3-4, 4-2, 4-4, 4-8, 4-14, 4-15

Detail 6-6, 6-8, 6-28, 7-14
 Detail Record(s) 4-14
 Detection 10-5, 10-6, 10-7, 11-10, 11-15, 11-19, 12-4, 12-8, 12-16, 12-53
 Dictionary 7-2, 7-8, 7-11, 8-21
 Dictionary Look-up 7-1, 7-2, 7-11, 7-12
 Digital Transmission 12-4
 Digit Analysis 7-3
 Direct Access Storage Device(s) 2-6, 2-9, 2-11, 2-12, 2-14, 2-15, 2-17, 5-19
 Direct Address 7-1, 7-2, 7-11, 7-12
 Directory(ies) 1-2, 1-6, 1-8, 1-21, 5-21, 6-1, 6-44, 9-7, 9-9, 9-11, 9-12, 12-25
 12-26, 12-33, 12-34, 12-37, 12-42
 Directory Language 12-34
 Disc 2-1, 2-4, 2-9, 2-11, 2-15, 2-16, 2-17, 4-9
 Distributed Computer System 12-50
 Distributed Data 1-20
 Distributed Data Base(s) 11-21, 12-21, 12-24, 12-25, 12-28, 12-31, 12-41
 Distributed Directory 12-33
 Distributed Network 12-19, 12-20, 12-21
 Distributed Overflow 7-9
 Division 7-5, 8-12, 8-15, 8-16
 Domain(s) 8-3, 8-5, 8-6, 8-8, 8-13, 8-14, 8-16, 8-20, 9-2, 9-9, 11-11, 11-13, 11-14,
 11-15, 11-16
 Double Update(s) 10-8
 Drum 2-1, 2-4, 2-7, 2-8, 2-11, 2-15, 2-17, 4-9
 Dump(s) 10-2, 10-3, 10-4, 10-5, 11-13
 Duplicate Files 12-40

 Element 1-11, 1-13, 1-15, 1-18, 1-19, 3-1, 3-4, 4-1
 Encryption 11-18
 End-of-Data 4-1, 4-10, 5-4
 End-of-File 2-6, 4-12, 5-17
 End-of-Reel 2-6
 End-of-Space 4-1, 5-1
 Enterprise 1-2, 1-5, 1-8, 1-11, 1-13, 1-19, 6-1, 8-1, 8-3
 Entity 1-11, 1-13, 1-15, 1-18, 8-3, 8-7
 Entity-Set(s) 1-11, 1-13, 1-18, 8-3, 8-5, 8-7
 Error Control 12-2, 12-3
 Error Recovery 12-8, 12-9
 Exclusive Access 10-8, 12-38
 Exclusive Control 10-8, 10-9
 Extent(s) 1-19, 1-20
 Extraction 7-3

Field 1-11, 1-15, 1-18, 6-26, 12-33
 File 1-16, 1-18, 1-19, 1-20, 5-8, 5-9, 12-33
 File Limit(s) 4-9, 4-12, 5-21
 File Management Supervisor 10-9, 10-10
 File Management System 11-15
 File Segment 12-25
 File Structure 2-16
 File Transfer 12-17
 File Transfer Protocol 12-17, 12-18
 First-In First-Out 3-1, 4-4, 6-28, 6-38
 First Normal Form 8-21, 9-2, 9-3, 9-5
 Flagging 4-1, 5-1, 6-28
 Flow Control 12-2, 12-3, 12-13
 Foible 11-8
 Folding 7-3, 7-5
 Forest 6-12, 6-14
 FORTRAN 1-2
 Frame 2-4, 2-5
 Free Space 4-1, 4-10, 4-14, 5-15, 5-16, 5-22, 5-23, 7-8, 7-9
 Full Index 5-9

 General Comprehensive Operating Supervisor 10-3
 Geography 12-24
 Global View 8-1

 Hardware 2-1, 11-10, 12-1, 12-2, 12-21
 Hardware Failure 10-2
 Hashing 5-9, 7-1, 7-2, 7-3, 7-5, 7-6, 7-7, 7-11, 7-12
 Head 6-28, 6-30, 6-33, 6-37, 7-12
 Header 6-37, 12-4, 12-5, 12-7, 12-8, 12-9, 12-51
 Heterogeneous Network 12-2, 12-17, 12-18, 12-49, 12-50
 High Level Protocols 12-17
 Home Bucket 7-8, 7-11
 Homogeneous Network 12-2
 Host(s) 12-2, 12-3, 12-4, 12-5, 12-8, 12-11, 12-13, 12-26, 12-28
 12-31, 12-33, 12-45, 12-48, 12-49, 12-50
 Host Computer(s) 12-2
 Host Language(s) 1-2, 1-15

 Identification 11-6, 11-17
 Impersonation 11-9
 Impersonator 11-10

Incomplete Update(s) 10-9, 10-10
 Independence 1-1, 4-7, 5-13, 6-27, 8-1, 10-6, 10-7
 Index 5-8, 5-9, 5-10, 5-13, 5-15, 5-17, 5-19, 5-20, 5-23, 6-27, 6-44, 6-51, 6-52,
 6-53, 7-1, 7-2, 7-10, 7-11, 7-12
 Indexed Sequential Access 5-8, 5-9, 5-22
 Indexed Sequential Access Method 6-53
 Indexed Sequential File(s) 5-10, 5-13, 5-16, 5-20, 5-21, 5-22, 7-1, 7-10
 Indexed Sequential Processor 4-14, 5-22, 10-3, 10-4, 10-5, 10-7
 Information 1-1, 1-6, 1-11, 1-15, 1-16, 1-20, 1-21, 6-1, 11-2, 11-18, 11-19, 12-21
 Information Management System 1-2, 8-20
 Input/Output 2-1
 Input/Output Control System 1-1
 Inquiry 12-35
 Insertion(s) 3-4, 4-1, 4-2, 4-4, 4-10, 4-14, 5-1, 5-3, 5-15, 5-16, 6-17, 6-27, 6-30,
 6-37, 6-51, 6-52
 Insertion Anomaly 9-9
 Integrated Data Share 1-2, 1-15, 4-14, 4-15, 4-16, 6-6, 6-8, 6-37, 6-53, 7-12, 7-14,
 10-3, 10-4, 10-5, 10-7, 11-21, 11-22
 Integrity 1-1, 6-33, 10-1, 10-2, 10-4, 10-5, 10-6, 10-7, 10-8, 12-10, 12-41
 Interblock Gap 2-6
 Interface Message Processor 12-20, 12-48, 12-49
 Inverted List(s) 5-17, 7-12
 Inverted List File 5-17, 5-19
 Item 1-16, 1-18

 Job Control Language 12-18, 12-49
 Join 8-12, 8-14, 8-15, 8-16
 Journal(s) 4-14, 5-22, 10-3, 10-4, 12-28
 Journalization 5-22, 12-26, 12-41

 Kernel 11-11, 11-13
 Key(s) 3-1, 3-4, 4-1, 4-7, 4-9, 4-16, 5-2, 5-4, 5-5, 5-7, 5-8, 5-9, 5-10, 5-13, 5-15,
 5-16, 5-19, 5-20, 6-8, 6-17, 6-19, 6-27, 6-44, 6-46, 6-49, 7-1, 7-2, 7-3,
 7-5, 7-6, 7-11, 7-12, 8-7, 9-9, 11-14, 11-18, 11-22, 11-23
 Knotted List 7-12

 Languages 1-2
 Last-In-First-Out 3-1, 4-2, 6-38
 Latency Time 2-14, 2-15

Libraries 2-17, 5-19
 Linear List 4-2, 4-4, 4-7, 5-2, 5-5
 Linear Relation 3-1
 Link 9-7, 12-13
 Linkage 6-8, 6-30, 6-33, 6-44, 7-14
 Linked List(s) 3-4, 4-15, 4-16, 5-7, 5-19
 Load Sharing 12-1, 12-53
 Lock(s) 11-23
 Lock Out 11-19, 11-20
 LOGGER 12-49
 Logical File 12-34
 Logical Ordering 3-1, 3-4, 4-14, 4-15
 Logical Organization 1-5, 1-11, 1-16, 1-18, 1-19, 1-20, 3-1, 3-4, 4-1, 4-7, 4-9, 6-1
 Logical Relationship(s) 6-12, 6-25, 6-28, 6-44, 6-50
 Logical Structure(s) 6-1, 6-9, 12-21
 Log-On 11-16
 Loop 6-22
 Lost Update 10-8
 Low Level Protocols 12-13, 12-16, 12-17

 Machine Address 6-26, 6-27
 Magnetic Tape 2-1, 2-4, 2-5, 2-6, 2-11, 2-14, 2-15, 2-16, 2-17, 4-9
 Main Frame 2-1
 Maintenance 12-25, 12-26, 12-28, 12-34
 Mass Storage System 2-11
 Master 6-6, 6-8, 6-28
 Master Record 4-14
 Master State 11-11
 Mathematical Model 11-2, 11-3
 Mechanical Identification 11-17
 Memory Management 11-10, 11-11
 Memory Protection 11-14
 Message(s) 12-2, 12-3, 12-5, 12-8, 12-9, 12-11, 12-19, 12-20, 12-26, 12-48,
 12-49, 12-50, 12-51, 12-54
 Message Editing 12-8, 12-9
 Message Queuing 12-8,
 Message Routers 12-45
 Message Switching 12-5, 12-6
 Microfilm 2-2
 Micro-processors 11-19
 Mid-Square 7-5

Minicomputer(s) 12-3, 12-4, 12-45, 12-50
 Model 1-6, 1-8, 1-13, 1-15, 1-16, 1-19, 9-7, 9-9
 Modem 12-4
 Monitor(s) 11-19, 11-20
 Multics 11-13, 11-14, 12-42, 12-44
 Multikey(ed) File(s) 5-9
 Multiplexing 12-51
 Multiplexor 12-7, 12-53

 Network 1-20, 6-20, 8-1, 8-21, 9-1, 9-2, 11-21, 12-1, 12-2, 12-3, 12-4, 12-6,
 12-7, 12-8, 12-9, 12-10, 12-13, 12-21, 12-24, 12-26, 12-28, 12-31, 12-33,
 12-34, 12-35, 12-41, 12-45, 12-48, 12-49, 12-50, 12-52, 12-53, 12-54
 Network Accounting Program 12-49
 Network Command Language 12-49, 12-50

 Network Control Program 12-45, 12-49, 12-54
 Node(s) 6-9, 6-10, 6-12, 6-14, 6-17, 6-19, 6-20, 6-22, 6-50, 12-4, 12-5, 12-21,
 12-24, 12-45
 Normalization 9-1, 9-2, 9-3, 9-6, 9-9
 Normalized Relation 1-18, 8-5, 8-20, 9-1, 9-2

 Open Addressing 7-8, 7-9
 Optical Character Recognition 2-2
 Ordered Tree(s) 6-10, 6-12
 Oriented Tree(s) 6-10
 Overflow 1-5, 4-12, 5-15, 5-16, 5-17, 5-21, 5-22, 7-6, 7-7, 7-8, 7-10, 7-11, 7-12
 Overhead 1-20, 6-28, 8-20, 10-6, 10-7, 11-3, 11-15, 12-6, 12-7, 12-35, 12-36,
 12-39, 12-41

 Packet(s) 12-2, 12-3, 12-7, 12-8, 12-20, 12-45, 12-48, 12-49,
 12-51, 12-54
 Packet Switching 12-6, 12-7, 12-45, 12-48, 12-51, 12-52, 12-54
 Packing Density 7-7, 7-10
 Page(s) 4-15, 5-9, 5-22, 10-9
 Parameter Processing 11-11
 Parity Check(ing) 2-5
 Partial Copy 12-26, 12-28, 12-31
 Partial Index 5-9, 5-20, 9-11, 10-5
 Password(s) 11-8, 11-9, 11-16, 11-17, 12-31
 Penetration(s) 11-10, 11-17
 Penetrator 11-16, 11-18, 11-19, 11-20, 11-21
 Permission(s) 11-15, 11-23

Physical Description Language(s) 1-2, 1-5
 Physical Ordering 3-1, 3-4, 4-8, 4-14, 5-20, 5-21
 Physical Organization 1-5, 1-11, 1-19, 1-20, 3-4, 4-1, 4-7, 4-12, 6-1, 6-50
 Physical Storage 6-25
 Plex(es) 6-20, 6-21, 6-22, 6-25, 6-50, 6-51, 9-1, 9-6
 PL1 1-2
 Pointer(s) 3-4, 4-1, 4-2, 4-4, 4-7, 4-10, 4-15, 5-1, 5-2, 5-4, 5-5, 5-9, 5-15, 5-16,
 5-17, 6-19, 6-26, 6-27, 6-28, 6-30, 6-33, 6-37, 6-38, 6-44, 6-50, 6-51,
 6-52, 6-53, 7-6, 7-8, 7-9, 7-12, 9-7, 11-13
 Point-to-Point Media 12-4, 12-5
 Polynomial(s) 7-5
 Precedence 12-49, 12-54
 Primary Copy 12-26, 12-28
 Primary Index 5-9, 5-20
 Primary Key(s) 5-8, 5-16, 5-17, 5-19, 8-7, 9-3, 9-9
 Primary Segment 12-31
 Prime Chain 6-8
 Priority 12-5, 12-8
 Privacy 11-2
 Privacy Act 1974 11-1, 11-2
 Privacy File 11-22, 11-23
 Probe 5-13
 Processes 12-8
 Projection 8-12, 8-13, 8-15, 8-16
 Protected Control 10-9
 Protection 11-16, 11-17, 11-18, 12-41
 Protocol(s) 12-2, 12-5, 12-6, 12-11, 12-13, 12-16, 12-17, 12-18, 12-49, 12-50
 Prototype WWMCCS Intercomputer Network 12-48

 Queue(s) 3-1, 4-2, 4-4, 4-8, 4-14, 4-15

 Radiation 11-9
 Random Access 2-7, 2-17, 5-1, 5-2, 5-3, 5-4, 5-5, 5-7, 7-1, 7-7
 Random Structure(s) 6-1, 6-33, 6-51, 6-52, 7-1, 7-10, 7-11, 7-12
 Read/Write Head 2-4, 2-7, 2-8, 2-9, 2-11, 2-14
 Real-time 12-25
 Record(s) 1-16, 1-18, 1-20, 2-6, 4-9, 4-10, 4-16, 5-8, 5-10, 5-15, 5-16, 5-17,
 6-26, 6-27, 6-28, 6-30, 6-33, 8-3, 8-7, 8-8, 12-33
 Record Based Sublanguage 8-9
 Record Control Word 5-17
 Record Identifier 6-26
 Record Journal 10-3, 10-4, 10-5

Recovery 6-51, 6-52, 10-4, 10-6, 10-9, 11-14, 11-21, 12-36
 Redundancy 6-50, 9-2, 12-45
 Relation(s) 1-15, 6-3, 6-9, 6-10, 8-3, 8-5, 8-6, 8-7, 8-8, 8-9, 8-13, 8-14, 8-15,
 8-17, 8-18, 8-19, 8-20, 9-1, 9-2, 9-7, 9-9, 9-11, 10-1
 Relational Calculus 8-18
 Relational Data Base 8-3, 8-8, 8-9, 8-14, 8-15, 8-18, 8-19, 8-20
 Relational Model 9-1
 Relational Structure(s) 1-15, 1-19, 8-1, 8-8, 8-20, 9-1
 Relational Sublanguage(s) 8-9, 8-12
 Relationship(s) 9-7
 Relative Address 5-9, 6-26, 6-27, 7-1
 Relative Sequence Number 5-2, 5-13
 Reliability 10-1, 12-1, 12-10, 12-26, 12-28, 12-30, 12-41, 12-54
 Remote Batch 12-17
 Remote Computer Concentrator 12-52
 Remote Job Entry 12-1, 12-2, 12-17, 12-18
 Repeating Group 1-15
 Request for Quotation 12-51
 Resource Allocation 11-10, 12-51
 Restart 10-4, 10-5
 Restoration 5-22
 Restricted Deque 4-4, 4-15
 Retrieval 1-8, 3-4, 4-16, 5-1, 5-8, 6-8, 6-46, 6-49, 12-34, 12-36, 12-38
 Ring(s) 6-37, 6-38, 6-50, 6-53, 7-12
 Ring Interface 12-10, 12-50
 Ring Network 12-19, 12-20, 12-50, 12-51
 Risk Analysis 11-4, 11-21
 Root 6-9, 6-10, 6-12, 6-14, 6-17, 6-19, 12-42
 Rotational Delay 2-14
 Routing 12-2, 12-3, 12-54

 Sabotage 11-8
 Satellite 12-52
 Scanning 4-7, 4-8, 5-7, 5-10, 5-13
 Schema(s) 1-11, 1-15, 1-16, 1-19, 6-1, 6-3, 6-6, 6-25, 6-46, 9-2, 9-5, 9-6, 9-11
 Secondary Index 5-9
 Secondary Key 5-8, 5-9, 5-19
 Secondary Storage 2-4
 Second Normal Form 9-9, 9-11
 Sector(s) 2-8, 2-9, 4-9, 5-22
 Security 11-1, 11-2, 11-3, 11-8, 11-10, 11-17, 11-21, 12-33

Security Breach(es) 11-8, 11-10, 11-13, 11-14, 11-19
 Security Checks 11-11
 Security Compromise(s) 11-8, 11-16, 11-19
 Security Control(s) 11-2, 11-3, 11-4, 11-11, 11-13, 11-15, 11-16, 11-19, 11-21, 11-22
 Security Lock(s) 11-22
 Security Parameter 11-20
 Seek Address 6-26
 Seek Time 2-14, 2-15
 Segment(s) 1-18, 12-25, 12-26, 12-28, 12-31, 12-42, 12-44
 Self-Contained Language(s) 1-2, 1-5
 Sensitivity of Data 11-3
 Sequential Access 2-4, 4-1, 4-2, 4-7, 4-8, 4-12, 4-16, 5-3, 5-5, 5-7, 5-10, 5-13
 Sequential File(s) 2-17
 Sequential Processing 2-7
 Serial Accessing 4-2, 4-4
 Serial Structure(s) 3-1, 3-4, 4-1, 4-2, 4-7, 4-8, 4-9, 4-10, 4-12, 5-1, 5-2, 5-3,
 5-4, 5-5, 5-7, 5-8, 5-19, 5-20, 5-21, 6-50, 6-52
 Serial Technique(s) 4-2
 Set(s) 1-19, 6-46
 Shifting 7-3, 7-5
 Simple Map 6-3
 Simple Relationship(s) 6-21
 Slave State 11-11
 Software Control 11-16
 Software Failure 10-2
 Sorting Technique(s) 3-1, 3-4
 Splitting 4-10, 5-15, 6-17, 8-13, 9-5, 9-6, 9-9, 9-11, 12-25
 Stack(s) 3-1, 4-2, 4-4, 4-8, 4-14
 Star Network 12-20
 Status 12-38, 12-40
 Store-and-Forward Procedure 12-5, 12-6, 12-48
 Strike-Over Mask 11-16
 Subcell 2-11
 Submodel 1-6, 1-8, 1-16
 Subnetwork 12-1, 12-2, 12-3, 12-4, 12-5, 12-8, 12-9, 12-11, 12-13, 12-16
 Subschema(s) 1-11, 1-16
 Subtree(s) 6-9, 6-10, 6-14, 9-3
 Surveillance Techniques 11-19
 Switches 12-2, 12-5, 12-6, 12-7, 12-8, 12-10, 12-13, 12-16
 Switching Centers 12-5
 Symbolic Address 5-9

Synchronized Update 12-39
 System Analyst(s) 1-1, 1-5, 1-6, 6-1
 System Control Processor 2-12

 Table(s) 8-3, 8-5, 8-8, 8-20, 11-17
 Tapping 11-9
 Teleconferences 12-33
 Teletype 12-2
 TELEX Switching System 12-53, 12-54
 TELNET 12-49
 Terabit Memory System 2-11, 2-12, 2-15, 2-16, 2-17
 Terminal(s) 12-2
 Terminal interface Processor 12-45
 Thesaurus 8-21, 9-1, 9-12, 12-33
 Third Normal Form 9-9, 9-11
 Threats 11-4, 11-8
 Timeout(s) 12-9
 Timesharing 4-16, 11-16, 11-23, 12-1, 12-19, 12-42, 12-52
 Track(s) 2-5, 2-7, 2-8, 2-9, 2-11, 4-9, 5-13, 5-15
 Transaction Driven System 4-14, 4-16
 Transaction Journal 10-3, 10-5
 Transaction Processing 12-40
 Transfer Rates 2-14, 2-15
 Transfer Time 2-14
 Transformation(s) 7-2, 7-6, 7-7, 11-17
 Transmission 12-3, 12-6, 12-7, 12-8, 12-9, 12-24, 12-35, 12-37,
 12-45, 12-53, 12-54
 Transmission Delays 12-10
 Transmission Errors 12-3, 12-6, 12-16
 Tree(s) 6-9, 6-10, 6-12, 6-17, 6-19, 6-20, 6-22, 6-25, 6-50, 6-51, 9-6, 12-42
 Trie 6-19
 Tuple(s) 8-2, 8-7, 8-13, 10-1

 Unauthorized Access 11-1, 11-16, 11-17, 12-33
 Unauthorized Hardware Components 11-10
 Unnormalized Relation 1-19
 Update(s) 4-7, 5-22, 10-4, 10-5, 10-7, 10-8, 10-9, 10-10, 12-26, 12-28, 12-30,
 12-34, 12-35, 12-36, 12-37, 12-38, 12-39, 12-40, 12-41
 Update Anomaly 9-9
 User Host 12-9
 Userid 11-9, 11-16
 Utility Software 11-13
 Utilization Report(s) 10-6, 10-7

Validation 11-17

Validity 10-1, 10-6

Variable Length Elements 4-2, 5-7, 5-8

Variable Length Records 5-16, 5-17, 5-22

Vector(s) 1-15

Verification 10-4

Virtual Memory 11-13

Virtual Storage Access Method 5-22, 6-53

Work Factor 11-19

Work Load Sharing 12-18, 12-19, 12-49, 12-50

Worldwide Data Management System 10-7, 11-21, 11-23

Worldwide Military Command and Control System 12-48

Write Ring 2-4